Package: seriation (via r-universe)

October 7, 2026

Type Package

Title Infrastructure for Ordering Objects Using Seriation

Version 1.6.0

Date 2026-10-06

Description Infrastructure for ordering objects with an implementation of several seriation/sequencing/ordination techniques to reorder matrices, dissimilarity matrices, and dendrograms. Also provides (optimally) reordered heatmaps, color images and clustering visualizations like dissimilarity plots, and visual assessment of cluster tendency plots (VAT and iVAT). Hahsler et al (2008) <doi:10.18637/jss.v025.i03>.

Classification/ACM G.1.6, G.2.1, G.4

URL https://github.com/mhahsler/seriation, http://michael.hahsler.net/seriation/

BugReports https://github.com/mhahsler/seriation/issues

Depends R (>= 4.3.0)

Imports ca, cluster, colorspace, foreach, gclus, grDevices, grid, MASS, qap, registry, stats, TSP

Suggests cba, corrgram, corrplot, corrr, dbscan (>= 1.2.7), DendSer, dendextend, doParallel, DT, dplyr, GA, ggplot2, heatmaply, keras, Rtsne, scales, smacof, tensorflow, testthat (>= 3.0.0), umap, vegan, viridis, knitr, rmarkdown

Encoding UTF-8

Roxygen list(markdown = TRUE)

VignetteBuilder knitr

Config/testthat/edition 3

License GPL-3

Copyright The code in src/bbwrcg.f, src/arsa.f and src/bburcg.f are Copyright (C) 2005 M. Brusco, H.F. Koehn, and S. Stahl. All other code is Copyright (C) Michael Hahsler, Christian Buchta, and Kurt Hornik.

Collate 'AAA_check_installed.R' 'AAA_color_palette.R' 'AAA_defaults.R'
'AAA_map.R' 'AAA_parameters.R' 'AAA_registry_criterion.R'
'AAA_registry_seriate.R' 'AAA_seriation-package.R'
'Chameleon.R' 'Irish.R' 'Munsingen.R' 'Psych24.R'
'SupremeCourt.R' 'Townships.R' 'VAT.R' 'Wood.R' 'Zoo.R'
'bertinplot.R' 'criterion.R' 'criterion.array.R'
'criterion.dist.R' 'criterion.matrix.R' 'dissplot.R'
'get_order.R' 'ggVAT.R' 'ggbertinplot.R' 'ggdissplot.R'
'hmap.R' 'gghmap.R' 'pimage.R' 'ggpimage.R' 'grid_helpers.R'
'lines_and_ordered_data.R' 'lle.R' 'permute.R'
'register_DendSer.R' 'register_GA.R' 'register_optics.R'
'register_smacof.R' 'register_tsne.R' 'register_umap.R'
'register_vegan.R' 'reorder.hclust.R' 'robinson.R' 'ser_dist.R'
'ser_permutation.R' 'ser_permutation_vector.R'
'ser_permutation_vector2matrix.R' 'seriate.R' 'seriate.dist.R'
'seriate.matrix.R' 'seriate.array.R' 'seriate.data.frame.R'
'seriate.table.R' 'seriate_AOE.R' 'seriate_ARSA_Branch-Bound.R'
'seriate_TSP.R' 'seriate_BEA.R' 'seriate_BK.R' 'seriate_CA.R'
'seriate_GSA.R' 'seriate_HC.R' 'seriate_LLE.R' 'seriate_MDS.R'
'seriate_Mean.R' 'seriate_PCA.R' 'seriate_QAP.R'
'seriate_R2E.R' 'seriate_SGLS.R' 'seriate_SPIN.R'
'seriate_VAT.R' 'seriate_best.R' 'seriate_enumerate.R'
'seriate_heatmap.R' 'seriate_identity.R' 'seriate_random.R'
'seriate_reverse.R' 'seriate_spectral.R' 'uniscale.R'

Config/roxygen2/version 8.1.0

Repository https://mhahsler.r-universe.dev

Date/Publication 2026-10-07 16:25:09 UTC

RemoteUrl https://github.com/mhahsler/seriation

RemoteRef HEAD

RemoteSha 03d5bdd102b8898f4650e61d6a8ba59f35f11a83

Contents

bertinplot . . . 3
Chameleon . . . 7
create_lines_data . . . 8
criterion . . . 10
dissplot . . . 17
get_order . . . 23
hmap . . . 25
Irish . . . 28
is.robinson . . . 29
lle . . . 30
LS . . . 32
Munsingen . . . 33
palette . . . 34

permutation_vector2matrix . . . 36
permute . . . 37
pimage . . . 40
Psych24 . . . 46
register_DendSer . . . 47
register_GA . . . 49
register_optics . . . 51
register_smacof . . . 52
register_tsne . . . 53
register_umap . . . 55
register_vegan . . . 56
registry_for_criterion_methods . . . 57
registry_for_seriation_methods . . . 59
reorder.hclust . . . 61
ser_dist . . . 63
ser_permutation . . . 66
ser_permutation_vector . . . 67
seriate . . . 69
seriate_best . . . 81
SupremeCourt . . . 84
Townships . . . 85
uniscale . . . 86
VAT . . . 88
Wood . . . 90
Zoo . . . 91

Index **93**

`bertinplot` *Plot a Bertin Matrix*

Description

Plot a data matrix of cases and variables. Each value is represented by a symbol. Large values are highlighted. Note that Bertin arranges the cases horizontally and the variables as rows. The matrix can be rearranged using seriation techniques to make structure in the data visible (see Falguerolles et al 1997).

Usage

```
bertinplot(
  x,
  order = NULL,
  panel.function = panel.bars,
  highlight = TRUE,
  row_labels = TRUE,
  col_labels = TRUE,
  flip_axes = TRUE,
```

```
  ...
)

panel.bars(value, spacing, hl)

panel.circles(value, spacing, hl)

panel.rectangles(value, spacing, hl)

panel.squares(value, spacing, hl)

panel.tiles(value, spacing, hl)

panel.blocks(value, spacing, hl)

panel.lines(value, spacing, hl)

bertin_cut_line(x = NULL, y = NULL, col = "red")

ggbertinplot(
  x,
  order = NULL,
  geom = "bar",
  highlight = TRUE,
  row_labels = TRUE,
  col_labels = TRUE,
  flip_axes = TRUE,
  prop = FALSE,
  ...
)
```

Arguments

x	a data matrix. Note that following Bertin, columns are variables and rows are cases. This behavior can be reversed using reverse = TRUE in options.
order	an object of class ser_permutation or a seriation method name to rearrange x before plotting. If NULL, no rearrangement is performed.
panel.function	a function to produce the symbols. Currently available functions are panel.bars (default), panel.circles, panel.rectangles, panel.tiles and panel.lines. For circles and squares neg. values are represented by a dashed border. For blocks all blocks are the same size (can be used with shading = TRUE).
highlight	a logical scalar indicating whether to use highlighting. If TRUE, all variables with values greater than the variable-wise mean are highlighted. To control highlighting, also a logical matrix or a matrix with colors with the same dimensions as x can be supplied.
row_labels, col_labels	a logical indicating if row and column labels in x should be displayed. If NULL then labels are displayed if the x contains the appropriate dimname and the num-

ber of labels is 25 or less. A character vector of the appropriate length with labels can also be supplied.

`flip_axes` logical indicating whether to swap cases and variables in the plot. The default (`TRUE`) is to plot cases as columns and variables as rows.

`...` `ggbertinplot()`: further parameters are passed on to `ggpimage()`.

`bertinplot()`: further parameters can include:

- `xlab, ylab` labels (default: use labels from `x`).
- `spacing` relative space between symbols (default: 0.2).
- `shading` use gray shades to encode value instead of highlighting (default: `FALSE`).
- `shading.function` a function that accepts a single argument in range $[.1, .8]$ and returns a valid corresponding color (e.g., using `rgb()`).
- `frame` plot a grid to separate symbols (default: `FALSE`).
- `mar` margins (see `par()`).
- `gp_labels` gpar object for labels (see `gpar()`)
- `gp_panels` gpar object for panels (see `gpar()`).
- `newpage` a logical indicating whether to start the plot on a new page (see `grid.newpage()`).
- `pop` a logical indicating whether to pop the created viewports (see `pop.viewport()`)?

`value, spacing, hl` are used internally for the panel functions.

`col, y` and `x` in `bertin_cut_line()` are for adding a line to a `bertinplot()` (not ggplot2-based).

`geom` visualization type. Available ggplot2 geometries are: `"tile"`, `"rectangle"`, `"circle"`, `"line"`, `"bar"`, `"none"`.

`prop` logical; change the aspect ratio so cells in the image have a equal width and height.

Details

The plot is organized as a matrix of symbols. The symbols are drawn by a panel function, where all symbols of a row are drawn by one call of the function (using vectorization). The interface for the panel function is `panel.myfunction(value, spacing, hl)`. `value` is the vector of values for a row scaled between 0 and 1, `spacing` contains the relative space between symbols and `hl` is a logical vector indicating which symbol should be highlighted.

Cut lines can be added to an existing Bertin plot using `bertin_cut_line(x = NULL, y = NULL)`. `x` and `y` indicate where to draw the cut line between two columns/rows. If both `x` and `y` are specified then one can select a row/column and the other can select a range to draw a line which does only span a part of the row/column. It is important to call `bertinplot()` with the option `pop = FALSE`.

`bertinplot()` uses `grid.lines()`, `grid.rect()` and `grid.circle()` to draw the panels. For large matrices, the outline around rectangles may cover most of the image. Use the parameter `gp_panels = gpar(col = NA)` to remove the outline.

`ggbertinplot()` calls `ggpimage()` and all additional parameters are passed on.

Value

Nothing.

Author(s)

Michael Hahsler

References

de Falguerolles, A., Friedrich, F., Sawitzki, G. (1997): A Tribute to J. Bertin's Graphical Data Analysis. In: Proceedings of the SoftStat '97 (Advances in Statistical Software 6), 11–20.

See Also

Other plots: VAT(), dissplot(), hmap(), palette, pimage()

Examples

```
data("Irish")
scale_by_rank <- function(x) apply(x, 2, rank)
x <- scale_by_rank(Irish[,-6])

# Use the the sum of absolute rank differences
order <- c(
  seriate(dist(x, "minkowski", p = 1)),
  seriate(dist(t(x), "minkowski", p = 1))
)

# Plot
bertinplot(x, order)

# Some alternative displays
bertinplot(x, order, panel.function = panel.tiles, shading_col = bluered(100), highlight = FALSE)
bertinplot(x, order, panel.function = panel.circles, spacing = -.2)
bertinplot(x, order, panel.function = panel.rectangles)
bertinplot(x, order, panel.function = panel.lines)

# Plot with cut lines (we manually set the order here)
order <- ser_permutation(c(6L, 9L, 29L, 10L, 32L, 22L, 2L, 35L,
  24L, 30L, 33L, 25L, 37L, 36L, 8L, 27L, 4L, 39L, 3L, 40L, 38L,
  1L, 31L, 34L, 28L, 23L, 5L, 11L, 7L, 41L, 13L, 26L, 17L, 15L,
  12L, 20L, 14L, 18L, 19L, 16L, 21L),
    c(4L, 2L, 1L, 6L, 7L, 8L, 5L, 3L))

bertinplot(x, order, pop=FALSE)
bertin_cut_line(, 4) ## horizontal line between rows 4 and 5
bertin_cut_line(, 7) ## separate "Right to Life" from the rest
bertin_cut_line(18, c(0, 4)) ## separate a block of large values (vertically)

# ggplot2-based plots
if (require("ggplot2")) {
  library(ggplot2)
```

```
  # Default plot uses bars and highlighting values larger than the mean
  ggbertinplot(x, order)

  # highlight values in the 4th quartile
  ggbertinplot(x, order, highlight = quantile(x, probs = .75))

  # Use different geoms. "none" lets the user specify their own geom.
  # Variables set are row, col and x (for the value).

  ggbertinplot(x, order, geom = "tile", prop = TRUE)
  ggbertinplot(x, order, geom = "rectangle")
  ggbertinplot(x, order, geom = "rectangle", prop = TRUE)
  ggbertinplot(x, order, geom = "circle")
  ggbertinplot(x, order, geom = "line")

  # Tiles with diverging color scale
  ggbertinplot(x, order, geom = "tile", prop = TRUE) +
    scale_fill_gradient2(midpoint = mean(x))

  # Custom geom (geom = "none"). Defined variables are row, col, and x for the value
  ggbertinplot(x, order, geom = "none", prop = FALSE) +
    geom_point(aes(x = col, y = row, size = x, color = x > 30), pch = 15) +
    scale_size(range = c(1, 10))

  # Use a ggplot2 theme with theme_set()
  old_theme <- theme_set(theme_minimal() +
      theme(panel.grid = element_blank())
    )
  ggbertinplot(x, order, geom = "bar")
  theme_set(old_theme)
}
```

`Chameleon` *2D Data Sets used for the CHAMELEON Clustering Algorithm*

Description

Several 2D data sets created to evaluate the CHAMELEON clustering algorithm in the paper by Karypis et al (1999).

Format

`chameleon_ds4`: The format is a 8,000 x 2 data.frame.

`chameleon_ds5`: The format is a 8,000 x 2 data.frame.

`chameleon_ds7`: The format is a 10,000 x 2 data.frame.

`chameleon_ds8`: The format is a 8,000 x 2 data.frame.

References

Karypis, G., EH. Han, V. Kumar (1999): CHAMELEON: A Hierarchical Clustering Algorithm Using Dynamic Modeling, *IEEE Computer,* **32**(8): 68–75. doi:10.1109/2.781637

See Also

Other data: `Irish`, `Munsingen`, `Psych24`, `SupremeCourt`, `Townships`, `Wood`, `Zoo`, `create_lines_data()`, `is.robinson()`

Examples

```
data(Chameleon)

plot(chameleon_ds4, cex = .1)
plot(chameleon_ds5, cex = .1)
plot(chameleon_ds7, cex = .1)
plot(chameleon_ds8, cex = .1)
```

---

create_lines_data *Create Simulated Data for Seriation Evaluation*

Description

Several functions to create simulated data to evaluate different aspects of seriation algorithms and criterion functions.

Usage

```
create_lines_data(n = 250)

create_ordered_data(
  n = 250,
  k = 2,
  size = NULL,
  spacing = 6,
  path = "linear",
  sd1 = 1,
  sd2 = 0
)
```

Arguments

`n`	number of data points to create.
`k`	number of Gaussian components.
`size`	relative size (number of points) of components (length of k). If `NULL` then all components have the same size.

spacing	space between the centers of components. The default of 6 means that the components will barely touch at ds1 = 1 (3 standard deviations for each Gaussian component).
path	Are the components arranged along a "linear" or "circular" path?
sd1	variation in the direction along the components. A value greater than one means the components are mixing.
sd2	variation perpendicular to the direction along the components. A value greater than 0 will introduce anti-Robinson violation events.

Details

create_lines_data() recreates the lines data set used in for iVAT() in Havens and Bezdek (2012).

create_ordered_data() (Hahsler et al, 2021) is a versatile function which creates "orderable" 2D data using Gaussian components along a linear or circular path. The components are equally spaced (spacing) along the path. The default spacing of 6 ensures that 2 adjacent components with a standard deviation of one along the direction of the path will barely touch. The standard deviation along the path is set by sd1. The standard deviation perpendicular to the path is set by sd2. A value larger than zero will result in the data not being perfectly orderable (i.e., the resulting distance matrix will not be a perfect pre-anti-Robinson matrix and contain anti-Robinson violation events after seriation). Note that a circular path always creates anti-Robinson violation since the circle has to be broken at some point to create a linear order.

Value

a data.frame with the created data.

Author(s)

Michael Hahsler

References

Havens, T.C. and Bezdek, J.C. (2012): An Efficient Formulation of the Improved Visual Assessment of Cluster Tendency (iVAT) Algorithm, *IEEE Transactions on Knowledge and Data Engineering,* **24**(5), 813–822.

Michael Hahsler, Christian Buchta and Kurt Hornik (2021). seriation: Infrastructure for Ordering Objects Using Seriation. R package version 1.3.2. https://github.com/mhahsler/seriation

See Also

seriate(), criterion(), iVAT().

Other data: Chameleon, Irish, Munsingen, Psych24, SupremeCourt, Townships, Wood, Zoo, is.robinson()

Examples

```
## lines data set from Havens and Bezdek (2011)
x <- create_lines_data(100)
plot(x, xlim = c(-5, 5), ylim = c(-3, 3), cex = .2, col = attr(x, "id"))
d <- dist(x)
pimage(d, seriate(d, "OLO_single"), col = bluered(100, bias = .5), key = TRUE)

## create_ordered_data can produce many types of "orderable" data

## perfect pre-Anti-Robinson matrix (with a single components)
x <- create_ordered_data(100, k = 1)
plot(x, cex = .2, col = attr(x, "id"))
d <- dist(x)
pimage(d, seriate(d, "MDS"), col = bluered(100, bias=.5), key = TRUE)

## separated components
x <- create_ordered_data(100, k = 5)
plot(x, cex =.2, col = attr(x, "id"))
d <- dist(x)
pimage(d, seriate(d, "MDS"), col = bluered(100, bias = .5), key = TRUE)

## overlapping components
x <- create_ordered_data(100, k = 5, sd1 = 2)
plot(x, cex = .2, col = attr(x, "id"))
d <- dist(x)
pimage(d, seriate(d, "MDS"), col = bluered(100, bias = .5), key = TRUE)

## introduce anti-Robinson violations (a non-zero y value)
x <- create_ordered_data(100, k = 5, sd1 = 2, sd2 = 5)
plot(x, cex = .2, col = attr(x, "id"))
d <- dist(x)
pimage(d, seriate(d, "MDS"), col = bluered(100, bias = .5), key = TRUE)

## circular path (has always violations)
x <- create_ordered_data(100, k = 5, path = "circular", sd1 = 2)
plot(x, cex = .2, col = attr(x, "id"))
d <- dist(x)
pimage(d, seriate(d, "OLO"), col = bluered(100, bias = .5), key = TRUE)

## circular path (with more violations violations)
x <- create_ordered_data(100, k = 5, path = "circular", sd1 = 2, sd2 = 1)
plot(x, cex=.2, col = attr(x, "id"))
d <- dist(x)
pimage(d, seriate(d, "OLO"), col = bluered(100, bias = .5), key = TRUE)
```

`criterion` *Criterion for a Loss/Merit Function for Data Given a Permutation*

Description

Compute the value for different loss functions L and merit function M for data given a permutation. A list with all methods and the available parameters is available here.

Usage

```
criterion(x, order = NULL, method = NULL, force_loss = FALSE, ...)

## S3 method for class 'array'
criterion(x, order = NULL, method = NULL, force_loss = FALSE, ...)

## S3 method for class 'dist'
criterion(x, order = NULL, method = NULL, force_loss = FALSE, ...)

## S3 method for class 'matrix'
criterion(x, order = NULL, method = NULL, force_loss = FALSE, ...)

## S3 method for class 'data.frame'
criterion(x, order = NULL, method = NULL, force_loss = FALSE, ...)

## S3 method for class 'table'
criterion(x, order = NULL, method = NULL, force_loss = FALSE, ...)
```

Arguments

x	an object of class dist or a matrix (currently no functions are implemented for array).
order	an object of class ser_permutation suitable for x. If NULL, the identity permutation is used.
method	a character vector with the names of the criteria to be employed (see list_criterion_methods()), or NULL (default) in which case all available criteria are used.
force_loss	logical; should merit function be converted into loss functions by multiplying with -1?
...	additional parameters passed on to the criterion method.

Details

Criteria for distance matrices (dist)

For a symmetric dissimilarity matrix D with elements $d(i, j)$ where $i, j = 1 \dots n$, the aim is generally to place low distance values close to the diagonal. The following criteria to judge the quality of a certain permutation of the objects in a dissimilarity matrix are currently implemented (for a more detailed description and an experimental comparison see Hahsler (2017)):

- **Gradient measures:** "Gradient_raw", "Gradient_weighted" (Hubert et al, 2001)

  A symmetric dissimilarity matrix where the values in all rows and columns only increase when moving away from the main diagonal is called a perfect *anti-Robinson matrix* (Robinson

1951). A suitable merit measure which quantifies the divergence of a matrix from the anti-Robinson form is

$$M(D) = \sum_{i=1}^{n} \sum_{i<k<j} f(d_{ij}, d_{ik}) + \sum_{i<k<j} f(d_{ij}, d_{kj})$$

where $f(.,.)$ is a function which defines how a violation or satisfaction of a gradient condition for an object triple (O_i, O_k, O_j) is counted.

Hubert et al (2001) suggest two functions. The first function is given by:

$$f(z, y) = sign(y - z) = +1 \text{ if } z < y;\ 0 \text{ if } z = y;\ \text{and } -1 \text{ if } z > y.$$

It results in raw number of triples satisfying the gradient constraints minus triples which violate the constraints.

The second function is defined as:

$$f(z, y) = |y - z| sign(y - z) = y - z$$

It weights the each satisfaction or violation by the difference by its magnitude given by the absolute difference between the values.

- **Anti-Robinson events:** `"AR_events"`, `"AR_deviations"` (Chen, 2002)
  `"AR_events"` counts the number of violations of the anti-Robinson form.

$$L(D) = \sum_{i=1}^{n} \sum_{i<k<j} f(d_{ik}, d_{ij}) + \sum_{i<k<j} f(d_{kj}, d_{ij})$$

with

$$f(z, y) = I(z, y) = 1 \text{ if } z < y \text{ and } 0 \text{ otherwise,}$$

where $I(.)$ is an indicator function returning 1 only for violations. Chen (2002) presented a formulation for an equivalent loss function and called the violations *anti-Robinson events.*

`"AR_deviations"`: Chen (2002) also introduced a weighted versions of the loss function by using

$$f(z, y) = |y - z| I(z, y)$$

which weights each violation by the deviation.

- **Relative generalized Anti-Robinson events:** `"RGAR"` (Tien et al, 2008)
  Counts Anti-Robinson events in a variable band (window specified by `w` defaults to the maximum of $n - 1$) around the main diagonal and normalizes by the maximum of possible events.

$$L(D) = 1/m \sum_{i=1}^{n} \sum_{(i-w)\leq j<k<i} I(d_{ij} < d_{ik}) + \sum_{i<j<k\leq(i+w))} I(d_{ij} > d_{ik})$$

where $m = (2/3 - n)w + nw^2 - 2/3w^3$, the maximal number of possible anti-Robinson events in the window. The window size w represents the number of neighboring objects (number of entries from the diagonal of the distance matrix) are considered. The window size is $2 \leq w < n$, where smaller values result in focusing on the local structure while larger values look at the global structure.

... parameters are:

- `w` window size. Default is to use a `pct` of 100% of n.
- `pct` and alternative specification of w as a percentage of n in $(0, 100]$.
- `relative` logical; can be set to `FALSE` to get the GAR, i.e., the absolute number of AR events in the window.

- **Banded anti-Robinson form criterion:** `"BAR"` (Earle and Hurley, 2015)

  Simplified measure for closeness to the anti-Robinson form in a band of size b with $1 <= b < n$ around the diagonal.

$$L(D) = \sum_{|i-j|<=b} (b + 1 - |i - j|)d_{ij}$$

  For $b = 1$ the measure reduces to the Hamiltonian path length. For $b = n - 1$ the measure is equivalent to ARc defined (see `register_DendSer()`).

  ... parameter is: `b` band size defaults to a band of 20% of n.

- **Hamiltonian path length:** `"Path_length"` (Caraux and Pinloche, 2005)

  The order of the objects in a dissimilarity matrix corresponds to a path through a graph where each node represents an object and is visited exactly once, i.e., a Hamilton path. The length of the path is defined as the sum of the edge weights, i.e., dissimilarities.

$$L(D) = \sum_{i=1}^{n-1} d_{i,i+1}$$

  The length of the Hamiltonian path is equal to the value of the minimal span loss function (as used by Chen 2002). Both notions are related to the *traveling salesperson problem (TSP).*

  If `order` is not unique or there are non-finite distance values `NA` is returned.

- **Lazy path length:** `"Lazy_path_length"` (Earl and Hurley, 2015)

  A weighted version of the Hamiltonian path criterion. This loss function postpones larger distances to later in the order (i.e., a lazy traveling sales person).

$$L(D) = \sum_{i=1}^{n-1} (n - i)d_{i,i+1}$$

  Earl and Hurley (2015) proposed this criterion for reordering in visualizations to concentrate on closer objects first.

- **Inertia criterion:** `"Inertia"` (Caraux and Pinloche, 2005)

  Measures the moment of the inertia of dissimilarity values around the diagonal as

$$M(D) = \sum_{i=1}^{n} \sum_{j=1}^{n} d(i, j)|i - j|^2$$

  $|i - j|$ is used as a measure for the distance to the diagonal and $d(i, j)$ gives the weight. This criterion gives higher weight to values farther away from the diagonal. It increases with quality.

- **Least squares criterion:** `"Least_squares"` (Caraux and Pinloche, 2005)
  The sum of squared differences between distances and the rank differences:

$$L(D) = \sum_{i=1}^{n}\sum_{j=1}^{n}(d(i,j) - |i-j|)^2,$$

  where $d(i,j)$ is an element of the dissimilarity matrix D and $|i-j|$ is the rank difference between the objects.

  Note that if Euclidean distance is used to calculate D from a data matrix X, the order of the elements in X by projecting them on the first principal component of X minimizes this criterion. The least squares criterion is related to *unidimensional scaling.*

- **Linear Seriation Criterion:** `"LS"` (Hubert and Schultz, 1976)
  Weights the distances with the absolute rank differences.

$$L(D)\sum_{i,j=1}^{n} d(i,j)|i-j|$$

- **2-Sum Criterion:** `"2SUM"` (Barnard, Pothen and Simon, 1993)
  The 2-Sum loss criterion multiplies the similarity between objects with the squared rank differences.

$$L(D)\sum_{i,j=1}^{n} 1/(1+d(i,j))(i-j)^2,$$

  where $s(i,j) = 1/(1+d(i,j))$ represents the similarity between objects i and j.

- **Absolute Spearman Correlation** `"Rho"`
  The absolute value of the Spearman rank correlation between the original distances and the rank differences in the order.

- **Matrix measures:** `"ME"`, `"Moore_stress"`, `"Neumann_stress"`
  These criteria are defined on general matrices (see below for definitions). The dissimilarity matrix is first converted into a similarity matrix using $S = 1/(1+D)$. If a different transformation is required, then perform the transformation first and supply a matrix instead of a dist object.

Criteria for matrices (matrix)

For a general matrix $X = x_{ij}$, $i = 1\ldots n$ and $j = 1\ldots m$, currently the following loss/merit functions are implemented:

- **Measure of Effectiveness:** `"ME"` (McCormick, 1972).
  The measure of effectiveness (ME) for matrix X, is defined as

$$M(X) = 1/2\sum_{i=1}^{n}\sum_{j=1}^{m} x_{i,j}(x_{i,j-1} + x_{i,j+1} + x_{i-1,j} + x_{i+1,j})$$

  with, by convention

$$x_{0,j} = x_{m+1,j} = x_{i,0} = x_{i,n+1} = 0.$$

ME is a merit measure, i.e. a higher ME indicates a better arrangement. Maximizing ME is the objective of the bond energy algorithm (BEA). ME is not defined for matrices with negative values. NA is returned in this case.

- **Weighted correlation coefficient:** "Cor_R" (Deutsch and Martin, 1971)

  Developed as the Measure of Effectiveness for the Moment Ordering Algorithm. R is a merit measure normalized so that its value always lies in $[-1, 1]$. For the special case of a square matrix $R = 1$ corresponds to only the main diagonal being filled, $R = 0$ to a random distribution of value throughout the array, and $R = -1$ to the opposite diagonal only being filled.

- **Matrix Stress:** "Moore_stress", "Neumann_stress" (Niermann, 2005)

  Stress measures the conciseness of the presentation of a matrix/table and can be seen as a purity function which compares the values in a matrix/table with its neighbors. The stress measure used here is computed as the sum of squared distances of each matrix entry from its adjacent entries.

$$L(X) = \sum_{i=1}^{n} \sum_{j=1}^{m} \sigma_{ij}$$

  The following types of neighborhoods are available:

  – Moore: comprises the eight adjacent entries.

$$\sigma_{ij} = \sum_{k=\max(1,i-1)}^{\min(n,i+1)} \sum_{l=\max(1,j-1)}^{\min(m,j+1)} (x_{ij} - x_{kl})^2$$

  – Neumann: comprises the four adjacent entries.

$$\sigma_{ij} = \sum_{k=\max(1,i-1)}^{\min(n,i+1)} (x_{ij} - x_{kj})^2 + \sum_{l=\max(1,j-1)}^{\min(m,j+1)} (x_{ij} - x_{il})^2$$

  The major difference between the Moore and the Neumann neighborhood is that for the later the contribution of row and column permutations to stress are independent and thus can be optimized independently.

Value

A named vector of real values.

Author(s)

Michael Hahsler

References

Barnard, S.T., A. Pothen, and H. D. Simon (1993): A Spectral Algorithm for Envelope Reduction of Sparse Matrices. *In Proceedings of the 1993 ACM/IEEE Conference on Supercomputing,* 493–502. Supercomputing '93. New York, NY, USA: ACM.

Caraux, G. and S. Pinloche (2005): Permutmatrix: A Graphical Environment to Arrange Gene Expression Profiles in Optimal Linear Order, *Bioinformatics,* **21**(7), 1280–1281.

Chen, C.-H. (2002): Generalized association plots: Information visualization via iteratively generated correlation matrices, *Statistica Sinica,* **12**(1), 7–29.

Deutsch, S.B. and J.J. Martin (1971): An ordering algorithm for analysis of data arrays. *Operational Research,* **19**(6), 1350–1362. doi:10.1287/opre.19.6.1350

Earle, D. and C.B. Hurley (2015): Advances in Dendrogram Seriation for Application to Visualization. *Journal of Computational and Graphical Statistics,* **24**(1), 1–25. doi:10.1080/10618600.2013.874295

Hahsler, M. (2017): An experimental comparison of seriation methods for one-mode two-way data. *European Journal of Operational Research,* **257**, 133–143. doi:10.1016/j.ejor.2016.08.066

Hubert, L. and J. Schultz (1976): Quadratic Assignment as a General Data Analysis Strategy. *British Journal of Mathematical and Statistical Psychology,* **29**(2). Blackwell Publishing Ltd. 190–241. doi:10.1111/j.20448317.1976.tb00714.x

Hubert, L., P. Arabie, and J. Meulman (2001): *Combinatorial Data Analysis: Optimization by Dynamic Programming.* Society for Industrial Mathematics. doi:10.1137/1.9780898718553

Niermann, S. (2005): Optimizing the Ordering of Tables With Evolutionary Computation, *The American Statistician,* **59**(1), 41–46. doi:10.1198/000313005X22770

McCormick, W.T., P.J. Schweitzer and T.W. White (1972): Problem decomposition and data reorganization by a clustering technique, *Operations Research,* **20**(5), 993-1009. doi:10.1287/opre.20.5.993

Robinson, W.S. (1951): A method for chronologically ordering archaeological deposits, *American Antiquity,* **16**, 293–301. doi:10.2307/276978

Tien, Y-J., Yun-Shien Lee, Han-Ming Wu and Chun-Houh Chen (2008): Methods for simultaneously identifying coherent local clusters with smooth global patterns in gene expression profiles, *BMC Bioinformatics,* **9**(155), 1–16. doi:10.1186/147121059155

See Also

Other criterion: registry_for_criterion_methods

Examples

```
## create random data and calculate distances
m <- matrix(runif(20),ncol=2)
d <- dist(m)

## get an order for rows (optimal for the least squares criterion)
o <- seriate(d, method = "MDS")
o

## compare the values for all available criteria
rbind(
    unordered = criterion(d),
    ordered = criterion(d, o)
)

## compare RGAR by window size (from local to global)
w <- 2:(nrow(m)-1)
RGAR <- sapply(w, FUN = function (w)
  criterion(d, o, method="RGAR", w = w))
plot(w, RGAR, type = "b", ylim = c(0,1),
```

```
  xlab = "Windows size (w)", main = "RGAR by window size")
```

`dissplot` *Dissimilarity Plot*

Description

Visualizes a dissimilarity matrix using seriation and matrix shading using the method developed by Hahsler and Hornik (2011). Entries with lower dissimilarities (higher similarity) are plotted darker. Dissimilarity plots can be used to uncover hidden structure in the data and judge cluster quality.

Usage

```
dissplot(
  x,
  labels = NULL,
  method = "spectral",
  control = NULL,
  lower_tri = TRUE,
  upper_tri = "average",
  diag = TRUE,
  cluster_labels = TRUE,
  cluster_lines = TRUE,
  reverse_columns = FALSE,
  options = NULL,
  ...
)

## S3 method for class 'reordered_cluster_dissimilarity_matrix'
plot(
  x,
  lower_tri = TRUE,
  upper_tri = "average",
  diag = TRUE,
  options = NULL,
  ...
)

## S3 method for class 'reordered_cluster_dissimilarity_matrix'
print(x, ...)

ggdissplot(
  x,
  labels = NULL,
  method = "spectral",
  control = NULL,
  lower_tri = TRUE,
```

```
  upper_tri = "average",
  diag = TRUE,
  cluster_labels = TRUE,
  cluster_lines = TRUE,
  reverse_columns = FALSE,
  ...
)
```

Arguments

x	an object of class dist.
labels	NULL or an integer vector of the same length as rows/columns in x indicating the cluster membership for each object in x as consecutive integers starting with one. The labels are used to reorder the matrix.
method	A single character string indicating the seriation method used to reorder the clusters (inter cluster seriation) as well as the objects within each cluster (intra cluster seriation). If different algorithms for inter and intra cluster seriation are required, method can be a list of two named elements (inter_cluster and intra_cluster each containing the name of the respective seriation method. Use list_seriation_methods() with kind = "dist" to find available algorithms. Set method to NA to plot the matrix as is (no or, if cluster labels are supplied, only coarse seriation). For intra cluster reordering with the special method "silhouette width" is available (for dissplot() only). Objects in clusters are then ordered by silhouette width (from silhouette plots). If no method is given, the default method of seriate.dist() is used. A third list element (named aggregation) can be added to control how inter cluster dissimilarities are computed from from the given dissimilarity matrix. The choices are "avg" (average pairwise dissimilarities; average-link), "min" (minimal pairwise dissimilarities; single-link), "max" (maximal pairwise dissimilarities; complete-link), and "Hausdorff" (pairs up each point from one cluster with the most similar point from the other cluster and then uses the largest dissimilarity of paired up points).
control	a list of control options passed on to the seriation algorithm. In case of two different seriation algorithms, control can contain a list of two named elements (inter_cluster and intra_cluster) containing each a list with the control options for the respective algorithm.
upper_tri, lower_tri, diag	a logical indicating whether to show the upper triangle, the lower triangle or the diagonal of the distance matrix. The string "average" can also be used to display within and between cluster averages in the two triangles.
cluster_labels	a logical indicating whether to display cluster labels in the plot.
cluster_lines	a logical indicating whether to draw lines to separate clusters.
reverse_columns	a logical indicating if the clusters are displayed on the diagonal from north-west to south-east (FALSE; default) or from north-east to south-west (TRUE).

`options` a list with options for plotting the matrix (`dissplot` only).

- `plot` a logical indicating if a plot should be produced. if `FALSE`, the returned object can be plotted later using the function `plot` which takes as the second argument a list of plotting options (see `options` below).
- `silhouettes` a logical indicating whether to include a silhouette plot (see Rousseeuw, 1987).
- `threshold` a numeric. If used, only plot distances below the threshold are displayed. Consider also using `zlim` for this purpose.
- `col` colors used for the image plot.
- `key` a logical indicating whether to place a color key below the plot.
- `zlim` range of values to display (defaults to range `x`).
- `axes` `"auto"` (default; enabled for less than 25 objects), `"y"` or `"none"`.
- `main` title for the plot.
- `newpage` a logical indicating whether to start plot on a new page (see `grid.newpage()`.
- `pop` a logical indicating whether to pop the created viewports? (see package **grid**)
- `gp`, `gp_lines`, `gp_labels` objects of class `gpar` containing graphical parameters for the plot lines and labels (see `gpar()`.

`...` `dissplot()`: further arguments are added to `options`. `ggdissplot()` further arguments are passed on to `ggpimage()`.

Details

The plot can also be used to visualize cluster quality (see Ling 1973). Objects belonging to the same cluster are displayed in consecutive order. The placement of clusters and the within cluster order is obtained by a seriation algorithm which tries to place large similarities/small dissimilarities close to the diagonal. Compact clusters are visible as dark squares (low dissimilarity) on the diagonal of the plot. Additionally, a Silhouette plot (Rousseeuw 1987) is added. This visualization is similar to CLUSION (see Strehl and Ghosh 2002), however, allows for using arbitrary seriating algorithms.

Note: Since `pimage()` uses **grid**, it should not be mixed with base R primitive plotting functions.

Value

`dissplot()` returns an invisible object of class `cluster_proximity_matrix` with the following elements:

`order` `NULL` or integer vector giving the order used to plot `x`.

`cluster_order` `NULL` or integer vector giving the order of the clusters as plotted.

`method` vector of character strings indicating the seriation methods used for plotting `x`.

`k` `NULL` or integer scalar giving the number of clusters generated.

`description` a `data.frame` containing information (label, size, average intra-cluster dissimilarity and the average silhouette) for the clusters as displayed in the plot (from top/left to bottom/right).

This object can be used for plotting via `plot(x, options = NULL, ...)`, where `x` is the object and `options` contains a list with plotting options (see above).

`ggdissplot()` returns a ggplot2 object representing the plot.

The plot description as an object of class `reordered_cluster_dissimilarity_matrix`.

Author(s)

Michael Hahsler

References

Hahsler, M. and Hornik, K. (2011): Dissimilarity plots: A visual exploration tool for partitional clustering. *Journal of Computational and Graphical Statistics,* **10**(2):335–354. doi:10.1198/jcgs.2010.09139

Ling, R.F. (1973): A computer generated aid for cluster analysis. *Communications of the ACM,* **16**(6), 355–361. doi:10.1145/362248.362263

Rousseeuw, P.J. (1987): Silhouettes: A graphical aid to the interpretation and validation of cluster analysis. *Journal of Computational and Applied Mathematics,* **20**(1), 53–65. doi:10.1016/0377-0427(87)901257

Strehl, A. and Ghosh, J. (2003): Relationship-based clustering and visualization for high-dimensional data mining. *INFORMS Journal on Computing,* **15**(2), 208–230. doi:10.1287/ijoc.15.2.208.14448

See Also

Other plots: `VAT()`, `bertinplot()`, `hmap()`, `palette`, `pimage()`

Examples

```
data("iris")

# shuffle rows
x_iris <- iris[sample(seq(nrow(iris))), -5]
d <- dist(x_iris)

# Plot original matrix
dissplot(d, method = NA)

# Plot reordered matrix using the nearest insertion algorithm (from tsp)
dissplot(d, method = "TSP", main = "Seriation (TSP)")

# Cluster iris with k-means and 3 clusters and reorder the dissimilarity matrix
l <- kmeans(x_iris, centers = 3)$cluster
dissplot(d, labels = l, main = "k-means")

# show only distances as lower triangle
dissplot(d, labels = l, main = "k-means", lower_tri = TRUE, upper_tri = FALSE)

# Use a grid layout to place several plots on a page
library("grid")
grid.newpage()
pushViewport(viewport(layout=grid.layout(nrow = 2, ncol = 2),
```

```
    gp = gpar(fontsize = 8)))
pushViewport(viewport(layout.pos.row = 1, layout.pos.col = 1))

# Visualize the clustering (using Spectral between clusters and MDS within)
res <- dissplot(d, l, method = list(inter_cluster = "Spectral",
                                    intra_cluster = "MDS"),
    main = "K-Means + Seriation", newpage = FALSE)

popViewport()
pushViewport(viewport(layout.pos.row = 1, layout.pos.col = 2))

# More visualization options. Note that we reuse the reordered object res!
# color: use 10 shades red-blue, biased towards small distances
plot(res, main = "K-Means + Seriation (red-blue + biased)",
    col= bluered(10, bias = .5), newpage = FALSE)

popViewport()
pushViewport(viewport(layout.pos.row = 2, layout.pos.col = 1))

# Threshold (using zlim) and cubic scale to highlight differences
plot(res, main = "K-Means + Seriation (cubic + threshold)",
    zlim = c(0, 2), col = grays(100, power = 3), newpage = FALSE)

popViewport()
pushViewport(viewport(layout.pos.row = 2, layout.pos.col = 2))

# Use gray scale with logistic transformation
plot(res, main = "K-Means + Seriation (logistic scale)",
  col = gray(
    plogis(seq(max(res$x_reordered), min(res$x_reordered), length.out = 100),
      location = 2, scale = 1/2, log.p = FALSE)
    ),
  newpage = FALSE)

popViewport(2)

# The reordered_cluster_dissimilarity_matrix object
res
names(res)

## ----------------------------------------------------------------------
## ggplot-based dissplot
if (require("ggplot2")) {

library("ggplot2")

# Plot original matrix
ggdissplot(d, method = NA)

# Plot seriated matrix
ggdissplot(d, method = "TSP") +
  labs(title = "Seriation (TSP)")
```

```
# Cluster iris with k-means and 3 clusters
l <- kmeans(x_iris, centers = 3)$cluster

ggdissplot(d, labels = l) +
  labs(title = "K-means + Seriation")

# show only lower triangle
ggdissplot(d, labels = l, lower_tri = TRUE, upper_tri = FALSE) +
  labs(title = "K-means + Seriation")

# No lines or cluster labels and add a label for the color key (fill)
ggdissplot(d, labels = l, cluster_lines = FALSE, cluster_labels = FALSE) +
  labs(title = "K-means + Seriation", fill = "Distances\n(Euclidean)")

# Diverging color palette with manual set midpoint and different seriation methods
ggdissplot(d, l, method = list(inter_cluster = "Spectral",
                               intra_cluster = "MDS")) +
  labs(title = "K-Means + Seriation", subtitle = "biased color scale") +
  scale_fill_gradient2(midpoint = median(d))

# Use manipulate scale using package scales
library("scales")

# Threshold (using limit and na.value) and cubic scale to highlight differences
cubic_dist_trans <- trans_new(
  name = "cubic",
  # note that we have to do the inverse transformation for distances
  transform = function(x) x^(1/3),
  inverse = function(x) x^3
)

ggdissplot(d, l, method = list(inter_cluster = "Spectral",
                               intra_cluster = "MDS")) +
  labs(title = "K-Means + Seriation", subtitle = "cubic + biased color scale") +
  scale_fill_gradient(low = "black", high = "white",
    limits = c(0,2), na.value = "white",
    transform = cubic_dist_trans)

# Use gray scale with logistic transformation
logis_2_.5_dist_trans <- trans_new(
  name = "Logistic transform (location, scale)",
  # note that we have to do the inverse transformation for distances
  transform = function(x) plogis(x, location = 2, scale = .5, log.p = FALSE),
  inverse = function(x) qlogis(x, location = 2, scale = .5, log.p = FALSE),
)

ggdissplot(d, l, method = list(inter_cluster = "Spectral",
                               intra_cluster = "MDS")) +
  labs(title = "K-Means + Seriation", subtitle = "logistic color scale") +
  scale_fill_gradient(low = "black", high = "white",
    transform = logis_2_.5_dist_trans,
    breaks = c(0, 1, 2, 3, 4))
}
```

get_order *Extracting Order Information from a Permutation Object*

Description

Method to get the order information from an object of class ser_permutation or ser_permutation_vector. Order information can be extracted as a permutation vector, a vector containing each object's rank or a permutation matrix.

Usage

```
get_order(x, ...)

## S3 method for class 'ser_permutation_vector'
get_order(x, ...)

## S3 method for class 'ser_permutation'
get_order(x, dim = 1, ...)

## S3 method for class 'hclust'
get_order(x, ...)

## S3 method for class 'dendrogram'
get_order(x, ...)

## S3 method for class 'integer'
get_order(x, ...)

## S3 method for class 'numeric'
get_order(x, ...)

get_rank(x, ...)

get_permutation_matrix(x, ...)
```

Arguments

x	an object of class ser_permutation or ser_permutation_vector.
...	further arguments are ignored for get_order(). For get_rank() and for get_permutation_matrix() the additional arguments are passed on to get_order() (e.g., as dim).
dim	order information for which dimension should be returned?

Details

get_order() returns the permutation as an integer vector which arranges the objects in the seriation order. That is, a vector with the index of the first, second, ..., n-th object in the order defined by the permutation. These permutation vectors can directly be used to reorder objects using subsetting

with "[". *Note:* In **seriation** we usually use these order-based permutation vectors. **Note on names:** While R's order() returns an unnamed vector, get_order() returns names (if available). The names are the object label corresponding to the index at that position. Therefore, the names in the order are in the order after the permutation.

get_rank() returns the seriation as an integer vector containing the rank/position for each objects after the permutation is applied. That is, a vector with the position of the first, second, ..., n-th object after permutation. Note: Use order() to convert ranks back to an order.

get_permutation_matrix() returns a $n \times n$ permutation matrix.

Value

Returns an integer permutation vector/a permutation matrix.

Author(s)

Michael Hahsler

See Also

Other permutation: permutation_vector2matrix(), permute(), reorder.hclust(), ser_dist(), ser_permutation(), ser_permutation_vector()

Examples

```
## create a random ser_permutation_vector
## Note that ser_permutation_vector is a single permutation vector
x <- structure(1:10, names = paste0("X", 1:10))
o <- sample(x)
o

p <- ser_permutation_vector(o)
p

get_order(p)
get_rank(p)
get_permutation_matrix(p)

## reorder objects using subsetting, the provided permute function or by
## multiplying the with the permutation matrix. We use here
x[get_order(p)]
permute(x, p)
drop(get_permutation_matrix(p) %*%  x)

## ser_permutation contains one permutation vector for each dimension
p2 <- ser_permutation(p, sample(5))
p2

get_order(p2, dim = 2)
get_rank(p2, dim = 2)
get_permutation_matrix(p2, dim = 2)
```

hmap *Plot Heat Map Reordered Using Seriation*

Description

Provides heatmaps reordered using several different seriation methods. This includes dendrogram based reordering with optimal leaf order and matrix seriation-based heat maps.

Usage

```
hmap(
  x,
  distfun = stats::dist,
  method = "OLO_complete",
  control = NULL,
  scale = c("none", "row", "column"),
  plot_margins = "auto",
  col = NULL,
  col_dist = grays(power = 2),
  row_labels = NULL,
  col_labels = NULL,
  ...
)

gghmap(
  x,
  distfun = stats::dist,
  method = "OLO_complete",
  control = NULL,
  scale = c("none", "row", "column"),
  prop = FALSE,
  ...
)
```

Arguments

x	a matrix or a dissimilarity matrix of class dist. If a dissimilarity matrix is used, then the distfun is ignored.
distfun	function used to compute the distance (dissimilarity) between both rows and columns. For gghmap(), this parameter is passed on in control.
method	a character strings indicating the used seriation algorithm (see seriate.dist()). If the method results in a dendrogram then stats::heatmap() is used to show the dendrograms, otherwise reordered distance matrices are shown instead.
control	a list of control options passed on to the seriation algorithm specified in method.
scale	character indicating if the values should be centered and scaled in either the row direction or the column direction, or none. Default is none.

plot_margins	character indicating what to show in the margins. Options are: "auto", "dendrogram", "distances", or "none".
col	a list of colors used.
col_dist	colors used for displaying distances.
row_labels, col_labels	a logical indicating if row and column labels in x should be displayed. If NULL then labels are displayed if the x contains the appropriate dimname and the number of labels is 25 or less. A character vector of the appropriate length with labels can also be supplied.
...	further arguments passed on to stats::heatmap().
prop	logical; change the aspect ratio so cells in the image have a equal width and height.

Details

For dendrogram based heat maps, the arguments are passed on to stats::heatmap() in **stats**. The following arguments for heatmap() cannot be used: margins, Rowv, Colv, hclustfun, reorderfun.

For seriation-based heat maps further arguments include:

- gp an object of class gpar containing graphical parameters (see gpar() in package **grid**).
- newpage a logical indicating whether to start plot on a new page
- prop a logical indicating whether the height and width of x should be plotted proportional to its dimensions.
- showdist Display seriated dissimilarity matrices? Values are "none", "both", "rows" or "columns".
- key logical; show a colorkey?
- key.lab Label plotted next to the color key.
- margins bottom and right-hand-side margins are calculated automatically or can be specifies as a vector of two numbers (in lines).
- zlim range of values displayed.
- col, col_dist color palettes used.

Value

An invisible list with elements:

rowInd, colInd	index permutation vectors.
reorder_method	name of the method used to reorder the matrix.

The list may contain additional elements (dendrograms, colors, etc).

Author(s)

Michael Hahsler

See Also

Other plots: VAT(), bertinplot(), dissplot(), palette, pimage()

Examples

```
data("Wood")

# Default heatmap does Euclidean distance, hierarchical clustering with
# complete-link and optimal leaf ordering. Note that the rows are
# ordered top-down in the seriation order (stats::heatmap orders in reverse)
hmap(Wood, main = "Wood (opt. leaf ordering)")
hmap(Wood, plot_margins = "distances", main = "Wood (opt. leaf ordering)")
hmap(Wood, plot_margins = "none", main = "Wood (opt. leaf ordering)")

# Heatmap with correlation-based distance, green-red color (greenred is
# predefined) and optimal leaf ordering and no row label
dist_cor <- function(x) as.dist(sqrt(1 - cor(t(x))))
hmap(Wood, distfun = dist_cor, col = greenred(100),
  main = "Wood (reordered by corr. between obs.)")

# Heatmap for distances
d <- dist(Wood)
hmap(d, main = "Wood (Euclidean distances)")

# order-based with dissimilarity matrices
hmap(Wood, method = "MDS_angle",
  col = greenred(100), col_dist = greens(100, power = 2),
  keylab = "norm. Expression", main = "Wood (reordered with distances)")

# Manually create a simple heatmap with pimage.
o <- seriate(Wood, method = "heatmap",
   control = list(dist_fun = dist, seriation_method = "OLO_ward"))
o

pimage(Wood, o)

# Note: method heatmap calculates reordered hclust objects which can be used
#       for many heatmap implementations like the standard implementation in
#       package stats.
heatmap(Wood, Rowv = as.dendrogram(o[[1]]), Colv = as.dendrogram(o[[2]]))

# ggplot 2 version does not support dendrograms in the margin (for now)
if (require("ggplot2")) {
  library("ggplot2")

  gghmap(Wood) + labs(title = "Wood", subtitle = "Optimal leaf ordering")

  gghmap(Wood, flip_axes = TRUE, prop = TRUE) +
    labs(title = "Wood", subtitle = "Optimal leaf ordering")

  dist_cor <- function(x) as.dist(sqrt(1 - cor(t(x))))
  gghmap(Wood, distfun = dist_cor) +
```

```
  labs(title = "Wood", subtitle = "Reordered by correlation between observations") +
  scale_fill_gradient2(low = "darkgreen", high = "red")

gghmap(d, prop = TRUE) +
  labs(title = "Wood", subtitle = "Euclidean distances, reordered")

# Note: the ggplot2-based version currently cannot show distance matrices
#      in the same plot.

# Manually seriate and plot as pimage.
o <- seriate(Wood, method = "heatmap", control = list(dist_fun = dist,
  seriation_method = "OLO_ward"))
o

ggpimage(Wood, o)
}
```

`Irish` *Irish Referendum Data Set*

Description

A data matrix containing the results of 8 referenda for 41 Irish communities used in Falguerolles et al (1997).

Format

The format is a 41 x 9 matrix. Two values are missing.

Details

Column 6 contains the size of the Electorate in 1992.

Source

The data was kindly provided by Guenter Sawitzki.

References

de Falguerolles, A., Friedrich, F., Sawitzki, G. (1997) A Tribute to J. Bertin's Graphical Data Analysis. In: *Proceedings of the SoftStat '97 (Advances in Statistical Software 6),* 11–20.

See Also

Other data: `Chameleon`, `Munsingen`, `Psych24`, `SupremeCourt`, `Townships`, `Wood`, `Zoo`, `create_lines_data()`, `is.robinson()`

Examples

```
data(Irish)
```

`is.robinson` *Create and Recognize Robinson and Pre-Robinson Matrices*

Description

Provides functions to create and recognize (anti) Robinson and pre-Robinson matrices. A (anti) Robinson matrix has strictly decreasing (increasing) values when moving away from the main diagonal. A pre-Robinson matrix is a matrix which can be transformed into a perfect Robinson matrix using simultaneous permutations of rows and columns.

Usage

```
is.robinson(x, anti = TRUE, pre = FALSE)

random.robinson(n, anti = TRUE, pre = FALSE, noise = 0)
```

Arguments

`x`	a symmetric, positive matrix or a dissimilarity matrix (a `dist` object).
`anti`	logical; check for anti Robinson structure? Note that for distances, anti Robinson structure is appropriate.
`pre`	logical; recognize/create pre-Robinson matrices.
`n`	number of objects.
`noise`	noise intensity between 0 and 1. Zero means no noise. Noise more than zero results in non-Robinson matrices.

Details

Note that the default matrices are anti Robinson matrices. This is done because distance matrices (the default in R) are typically anti Robinson matrices with values increasing when moving away from the diagonal.

Robinson matrices are recognized using the fact that they have zero anti Robinson events. For pre-Robinson matrices we use spectral seriation first since spectral seriation is guaranteed to perfectly reorder pre-Robinson matrices (see Laurent and Seminaroti, 2015).

Random pre-Robinson matrices are generated by reversing the process of unidimensional scaling. We randomly (uniform distribution with range $[0, 1]$) choose x coordinates for `n` points on a straight line and calculate the pairwise distances. For Robinson matrices, the points are sorted first according to x. For noise, y coordinates is added. The coordinates are chosen uniformly between 0 and `noise`, with `noise` between 0 and 1.

Value

A single logical value.

References

M. Laurent, M. Seminaroti (2015): The quadratic assignment problem is easy for Robinsonian matrices with Toeplitz structure, *Operations Research Letters* **43**(1), 103–109.

See Also

Other data: Chameleon, Irish, Munsingen, Psych24, SupremeCourt, Townships, Wood, Zoo, create_lines_data()

Examples

```
## create a perfect anti Robinson structure
m <- random.robinson(10)
pimage(m)

is.robinson(m)

## permute the structure to make it not Robinsonian. However,
## it is still pre-Robinson.
o <- sample(10)
m2 <- permute(m, ser_permutation(o,o))
pimage(m2)

is.robinson(m2)
is.robinson(m2, pre = TRUE)

## create a binary random Robinson matrix (not anti Robinson)
m3 <- random.robinson(10, anti = FALSE) > .7
pimage(m3)
is.robinson(m3, anti = FALSE)

## create matrices with noise (as distance matrices)
m4 <- as.dist(random.robinson(50, pre = FALSE, noise = .1))
pimage(m4)
criterion(m4, method = "AR")

m5 <- as.dist(random.robinson(50, pre = FALSE, noise = .5))
pimage(m5)
criterion(m5, method = "AR")
```

---

lle *Locally Linear Embedding (LLE)*

Description

Performs the non linear dimensionality reduction method locally linear embedding proposed in Roweis and Saul (2000).

Usage

```
lle(x, m, k, reg = 2)
```

Arguments

x	a matrix.
m	dimensions of the desired embedding.
k	number of neighbors.
reg	regularization method. 1, 2 and 3, by default 2. See details.

Details

LLE tries to find a lower-dimensional projection which preserves distances within local neighborhoods. This is done by (1) find for each object the k nearest neighbors, (2) construct the LLE weight matrix which represents each point as a linear combination of its neighborhood, and (2) perform partial eigenvalue decomposition to find the embedding.

The reg parameter allows the decision between different regularization methods. As one step of the LLE algorithm, the inverse of the Gram-matrix $G \in R^{kxk}$ has to be calculated. The rank of G equals m which is mostly smaller than k - this is why a regularization $G^{(i)} + r \cdot I$ should be performed. The calculation of regularization parameter r can be done using different methods:

- reg = 1: standardized sum of eigenvalues of G (Roweis and Saul; 2000)
- reg = 2 (default): trace of Gram-matrix divided by k (Grilli, 2007)
- reg = 3: constant value 3*10e-3

Value

a matrix of vector with the embedding.

Author(s)

Michael Hahsler (based on code by Holger Diedrich and Markus Abel)

References

Roweis, Sam T. and Saul, Lawrence K. (2000), Nonlinear Dimensionality Reduction by Locally Linear Embedding, *Science,* **290**(5500), 2323–2326. doi:10.1126/science.290.5500.2323

Grilli, Elisa (2007) Automated Local Linear Embedding with an application to microarray data, Dissertation thesis, University of Bologna. doi:10.6092/unibo/amsdottorato/380

See Also

Other helper: LS, uniscale()

Examples

```
data(iris)
x <- iris[, -5]

# project iris on 2 dimensions
conf <- lle(x, m = 2, k = 30)
conf

plot(conf, col = iris[, 5])

# project iris onto a single dimension
conf <- lle(x, m = 1, k = 30)
conf

plot_config(conf, col = iris[, 5], labels = FALSE)
```

LS *Neighborhood functions for Seriation Method SA*

Description

Definition of different local neighborhood functions for the method "SA" for seriate().

Usage

```
LS_swap(o, pos = sample.int(length(o), 2))

LS_insert(o, pos = sample.int(length(o), 2))

LS_reverse(o, pos = sample.int(length(o), 2))

LS_mixed(o, pos = sample.int(length(o), 2))
```

Arguments

o	an integer vector with the order
pos	random positions used for the local move.

Details

Local neighborhood functions are LS_insert, LS_swap, LS_reverse, and LS_mix (1/3 insertion, 1/3 swap and 1/3 reverse). Any neighborhood function can be defined.

Value

returns the new order vector representing the random neighbor.

See Also

Other helper: lle(), uniscale()

Munsingen *Hodson's Munsingen Data Set*

Description

This data set contains a grave times artifact incidence matrix for the Celtic Münsingen-Rain cemetery in Switzerland as provided by Hodson (1968) and published by Kendall 1971.

Format

A 59 x 70 0-1 matrix. Rows (graves) and columns (artifacts) are in the order determined by Hodson (1968).

References

Hodson, F.R. (1968). *The La Tene Cemetery at Münsingen-Rain,* Stämpfli, Bern.

Kendall, D.G. (1971): Seriation from abundance matrices. In: Hodson, F.R., Kendall, D.G. and Tautu, P., (Editors), *Mathematics in the Archaeological and Historical Sciences,* Edinburgh University Press, Edinburgh, 215–232.

See Also

Other data: Chameleon, Irish, Psych24, SupremeCourt, Townships, Wood, Zoo, create_lines_data(), is.robinson()

Examples

```
data("Munsingen")

## Seriation method after Kendall (1971)
## Kendall's square symmetric matrix S and SoS
S <- function(x, w = 1) {
  sij <- function(i , j) w * sum(pmin(x[i,], x[j,]))
  h <- nrow(x)
  r <- matrix(ncol = h, nrow =h)
  for(i in 1:h) for (j in 1:h)  r[i,j] <- sij(i,j)
  r
}

SoS <- function(x) S(S(x))

## Kendall's horse shoe (Hamiltonian arc)
horse_shoe_plot <- function(mds, sigma, threshold = mean(sigma), ...) {
    plot(mds, main = paste("Kendall's horse shoe with th =", threshold), ...)
    l <- which(sigma > threshold, arr.ind=TRUE)
```

```
    for(i in 1:nrow(l))  lines(rbind(mds[l[i,1],], mds[l[i,2],]))
}

## shuffle data
x <- Munsingen[sample(nrow(Munsingen)),]

## calculate matrix and do isoMDS (from package MASS)
sigma <- SoS(x)
library("MASS")
mds <- isoMDS(1/(1+sigma))$points

## plot Kendall's horse shoe
horse_shoe_plot(mds, sigma)

## find order using a TSP
library("TSP")
tour <- solve_TSP(insert_dummy(TSP(dist(mds)), label = "cut"),
    method = "2-opt", control = list(rep = 15))
tour <- cut_tour(tour, "cut")
lines(mds[tour,], col = "red", lwd = 2)

## create and plot order
order <- ser_permutation(tour, 1:ncol(x))
bertinplot(x, order, options= list(panel=panel.circles,
    rev = TRUE))

## compare criterion values
rbind(
    random = criterion(x),
    reordered = criterion(x, order),
    Hodson = criterion(Munsingen)
   )
```

---

palette *Different Useful Color Palettes*

Description

Defines several color palettes for `pimage()`, `dissplot()` and `hmap()`.

Usage

```
bluered(n = 100, bias = 1, power = 1, ...)

greenred(n = 100, bias = 1, power = 1, ...)

reds(n = 100, bias = 1, power = 1, ...)

blues(n = 100, bias = 1, power = 1, ...)
```

```
greens(n = 100, bias = 1, power = 1, ...)

greys(n = 100, bias = 1, power = 1, ...)

grays(n = 100, bias = 1, power = 1, ...)
```

Arguments

n	number of different colors produces.
bias	a positive number. Higher values give more widely spaced colors at the high end.
power	used to control how chroma and luminance is increased (1 = linear, 2 = quadratic, etc.)
...	further parameters are passed on to colorspace::sequential_hcl() or colorspace::diverging_hcl(

Details

The color palettes are created with colorspace::sequential_hcl() and colorspace::diverging_hcl().

The two sequential palettes are: reds() and grays() (or greys()).

The two diverging palettes are: bluered() and greenred().

Value

A vector with n colors.

Author(s)

Michael Hahsler

See Also

Other plots: VAT(), bertinplot(), dissplot(), hmap(), pimage()

Examples

```
m <- outer(1:10,1:10)
m

pimage(m)
pimage(m, col = greys(100, power = 2))
pimage(m, col = greys(100, bias = 2))
pimage(m, col = bluered(100))
pimage(m, col = bluered(100, power = .5))
pimage(m, col = bluered(100, bias = 2))
pimage(m - 25, col = greenred(20, bias = 2))

## choose your own color palettes
library(colorspace)
hcl_palettes(plot = TRUE)
```

```
## blues (with 20 shades)
pimage(m,
  col = colorspace::sequential_hcl(20, "Blues", rev = TRUE))
## blue to green (aka "Cork")
pimage(m,
  col = colorspace::diverging_hcl(100, "Cork"))
```

`permutation_vector2matrix` *Conversion Between Permutation Vector and Permutation Matrix*

Description

Converts between permutation vectors and matrices.

Usage

```
permutation_vector2matrix(x)

permutation_matrix2vector(x)
```

Arguments

x	A permutation vector (any object that can be converted into a permutation vector, e.g., a integer vector or a `hclust` object) or a matrix representing a permutation. Arguments are checked.

Value

- `permutation_vector2matrix()`: returns a permutation matrix.
- `permutation_matrix2vector()`: returns the permutation as a integer vector.

Author(s)

Michael Hahsler

See Also

Other permutation: `get_order()`, `permute()`, `reorder.hclust()`, `ser_dist()`, `ser_permutation()`, `ser_permutation_vector()`

Examples

```
## create a random permutation vector
pv <- structure(sample(5), names = paste0("X", 1:5))
pv

## convert into a permutation matrix
pm <- permutation_vector2matrix(pv)
pm

## convert back
permutation_matrix2vector(pm)
```

`permute` *Permute the Order in Various Objects*

Description

Provides the generic function and methods for permuting the order of various objects including vectors, lists, dendrograms (also `hclust` objects), the order of observations in a `dist` object, the rows and columns of a matrix or data.frame, and all dimensions of an array given a suitable ser_permutation object.

Usage

```
permute(x, order, ...)

## S3 method for class 'array'
permute(x, order, margin = NULL, ...)

## S3 method for class 'matrix'
permute(x, order, margin = NULL, ...)

## S3 method for class 'data.frame'
permute(x, order, margin = NULL, ...)

## S3 method for class 'table'
permute(x, order, margin = NULL, ...)

## S3 method for class 'numeric'
permute(x, order, ...)

## S3 method for class 'character'
permute(x, order, ...)

## S3 method for class 'list'
permute(x, order, ...)
```

```
## S3 method for class 'dist'
permute(x, order, ...)

## S3 method for class 'dendrogram'
permute(x, order, dist = NULL, ...)

## S3 method for class 'hclust'
permute(x, order, dist = NULL, ...)
```

Arguments

x	an object (a list, a vector, a dist object, a matrix, an array or any other object which provides dim and standard subsetting with "[").
order	an object of class ser_permutation which contains suitable permutation vectors for x. Alternatively, a character string with the name of a seriation method appropriate for x can be specified (see seriate()). This will perform seriation and permute x. The value TRUE will permute using the default seriation method.
...	if order is the name of a seriation method, then additional arguments are passed on to seriate().
margin	specifies the dimensions to be permuted as a vector with dimension indices. If NULL, order needs to contain a permutation for all dimensions. If a single margin is specified, then order can also contain a single permutation vector. margin are ignored.
dist	the distance matrix used to create the dendrogram. Only needed if order is the name of a seriation method.

Details

The permutation vectors in ser_permutation are suitable if the number of permutation vectors matches the number of dimensions of x and if the length of each permutation vector has the same length as the corresponding dimension of x.

For 1-dimensional/1-mode data (list, vector, dist), order can also be a single permutation vector of class ser_permutation_vector or data which can be automatically coerced to this class (e.g. a numeric vector).

For dendrogram and hclust, subtrees are rotated to represent the order best possible. If the order is not achieved perfectly then the user is warned. See also reorder.hclust() for reordering hclust objects.

Value

A permuted object of the same class as x.

Author(s)

Michael Hahsler

See Also

Other permutation: get_order(), permutation_vector2matrix(), reorder.hclust(), ser_dist(), ser_permutation(), ser_permutation_vector()

Examples

```
# List data types for permute
methods("permute")

# Permute matrix
m <- matrix(rnorm(10), 5, 2, dimnames = list(1:5, LETTERS[1:2]))
m

# Permute rows and columns
o <- ser_permutation(5:1, 2:1)
o

permute(m, o)

## permute only columns
permute(m, o, margin = 2)

## permute using PCA seriation
permute(m, "PCA")

## permute only rows using PCA
permute(m, "PCA", margin = 1)

# Permute data.frames using heatmap seriation (= hierarchical
#  clustering + optimal leaf ordering)
df <- as.data.frame(m)
permute(df, "Heatmap")

# Permute objects in a dist object
d <- dist(m)
d

permute(d, c(3, 2, 1, 4, 5))

permute(d, "Spectral")

# Permute a list
l <- list(a = 1:5, b = letters[1:3], c = 0)
l

permute(l, c(2, 3, 1))

# Permute to reorder dendrogram (see also reorder.hclust)
hc <- hclust(d)
plot(hc)

plot(permute(hc, 5:1))
```

```
plot(permute(hc, "OLO", dist = d))
plot(permute(hc, "GW", dist = d))
plot(permute(hc, "MDS", dist = d))
plot(permute(hc, "TSP", dist = d))
```

---

pimage *Permutation Image Plot*

Description

Provides methods for matrix shading, i.e., displaying a color image for matrix (including correlation matrices and data frames) and `dist` objects given an optional permutation. The plot arranges colored rectangles to represent the values in the matrix. This visualization is also know as a heatmap. Implementations based on the **grid** graphics engine and based n **ggplot2** are provided.

Usage

```
pimage(x, order = FALSE, ...)

## S3 method for class 'matrix'
pimage(
  x,
  order = FALSE,
  col = NULL,
  main = "",
  xlab = "",
  ylab = "",
  zlim = NULL,
  key = TRUE,
  keylab = "",
  symkey = TRUE,
  upper_tri = TRUE,
  lower_tri = TRUE,
  diag = TRUE,
  row_labels = NULL,
  col_labels = NULL,
  prop = isSymmetric(x),
  flip_axes = FALSE,
  reverse_columns = FALSE,
  ...,
  newpage = TRUE,
  pop = TRUE,
  gp = NULL
)

## S3 method for class 'table'
```

```
pimage(x, order = NULL, ...)

## S3 method for class 'data.frame'
pimage(x, order = NULL, ...)

## S3 method for class 'dist'
pimage(
  x,
  order = NULL,
  col = NULL,
  main = "",
  xlab = "",
  ylab = "",
  zlim = NULL,
  key = TRUE,
  keylab = "",
  symkey = TRUE,
  upper_tri = TRUE,
  lower_tri = TRUE,
  diag = TRUE,
  row_labels = NULL,
  col_labels = NULL,
  prop = TRUE,
  flip_axes = FALSE,
  reverse_columns = FALSE,
  ...,
  newpage = TRUE,
  pop = TRUE,
  gp = NULL
)

ggpimage(x, order = NULL, ...)

## S3 method for class 'matrix'
ggpimage(
  x,
  order = NULL,
  zlim = NULL,
  upper_tri = TRUE,
  lower_tri = TRUE,
  diag = TRUE,
  row_labels = NULL,
  col_labels = NULL,
  prop = isSymmetric(x),
  flip_axes = FALSE,
  reverse_columns = FALSE,
  ...
)
```

```
## S3 method for class 'dist'
ggpimage(
  x,
  order = NULL,
  zlim = NULL,
  upper_tri = TRUE,
  lower_tri = TRUE,
  diag = TRUE,
  row_labels = NULL,
  col_labels = NULL,
  prop = TRUE,
  flip_axes = FALSE,
  reverse_columns = FALSE,
  ...
)
```

Arguments

x	a matrix, a data.frame, or an object of class dist.
order	a logical where FALSE means no reordering and TRUE applies a permutation using the default seriation method for the type of x. Alternatively, the name of a seriation method or any object that can be coerced to class ser_permutation can be supplied.
...	if order is the name of a seriation method then further arguments are passed on to the seriation method, otherwise they are ignored.
col	a list of colors used. If NULL, a gray scale is used (for matrix larger values are displayed darker and for dist smaller distances are darker). For matrices containing logical data, black and white is used. For matrices containing negative values a symmetric diverging color palette is used.
main	plot title.
xlab, ylab	labels for the x and y axes.
zlim	vector with two elements giving the range (min, max) for representing the values in the matrix.
key	logical; add a color key? No key is available for logical matrices.
keylab	string plotted next to the color key.
symkey	logical; if x contains negative values, should the color palate be symmetric (zero is in the middle)?
upper_tri, lower_tri, diag	a logical indicating whether to show the upper triangle, the lower triangle or the diagonal of the (distance) matrix.
row_labels, col_labels	a logical indicating if row and column labels in x should be displayed. If NULL then labels are displayed if the x contains the appropriate dimname and the number of labels is 25 or less. A character vector of the appropriate length with labels can also be supplied.

prop	logical; change the aspect ratio so cells in the image have a equal width and height.
flip_axes	logical; exchange rows and columns for plotting.
reverse_columns	logical; revers the order of how the columns are displayed.
newpage, pop, gp	Start plot on a new page, pop the viewports after plotting, and use the supplied gpar object (see **grid**).

Details

Plots a matrix in its original row and column orientation (image in **stats** reverses the rows). This means, in a plot the columns become the x-coordinates and the rows the y-coordinates (in reverse order).

Grid-based plot: The viewports used for plotting are called: "plot", "image" and "colorkey". Use **grid** functions to manipulate the plots (see Examples section).

ggplot2-based plot: A ggplot2 object is returned. Colors, axis limits and other visual aspects can be added using standard ggplot2 functions (labs, scale_fill_continuous, labs, etc.).

Value

Nothing.

Author(s)

Christian Buchta and Michael Hahsler

See Also

Other plots: VAT(), bertinplot(), dissplot(), hmap(), palette

Examples

```
set.seed(1234)
data(iris)
x <- as.matrix(iris[sample(nrow(iris), 20) , -5])

pimage(x)

# Show all labels and flip axes, reverse columns, or change colors
pimage(x, prop = TRUE)
pimage(x, flip_axes = TRUE)
pimage(x, reverse_columns = TRUE)
pimage(x, col = grays(100))

# A matrix with positive and negative values
x_scaled <- scale(x)
pimage(x_scaled)

# Use reordering
pimage(x_scaled, order = TRUE)
```

```
pimage(x_scaled, order = "Heatmap")

## Example: Distance Matrix
# Show a reordered distance matrix (distances between rows).
# Dark means low distance. The aspect ratio is automatically fixed to 1:1
# using prop = TRUE
d <- dist(x)
pimage(d)
pimage(d, order = TRUE)

# Suppress the upper triangle and diagonal
pimage(d, order = TRUE, upper_tri = FALSE, diag = FALSE)

# Show only distances that are smaller than 2 using limits on z.
pimage(d, order = TRUE, zlim = c(0, 3))

## Example: Correlation Matrix
# we calculate correlation between rows and seriate the matrix
# and seriate by converting the correlations into distances.
# pimage reorders then rows and columns with c(o, o).
r <- cor(t(x))
o <- seriate(as.dist(sqrt(1 - r)))
pimage(r, order = c(o, o),
  upper_tri = FALSE, diag = FALSE,
  zlim = c(-1, 1),
  reverse_columns = TRUE,
  main = "Correlation matrix")

# Add to the plot using functions in package grid
# Note: pop = FALSE allows us to manipulate viewports
library("grid")
pimage(x, order = TRUE, pop = FALSE)

# available viewports are: "main", "colorkey", "plot", "image"
current.vpTree()

# Highlight cell 2/2 with a red arrow
# Note: columns are x and rows are y.
downViewport(name = "image")
grid.lines(x = c(1, 2), y = c(-1, 2), arrow = arrow(),
  default.units = "native", gp = gpar(col = "red", lwd = 3))

# add a red box around the first 4 rows of the 2nd column
grid.rect(x = 1 + .5 , y = 4 + .5, width = 1, height = 4,
  hjust = 0, vjust = 1,
  default.units = "native", gp = gpar(col = "red", lwd = 3, fill = NA))

## remove the viewports
popViewport(0)

## put several pimages on a page (use grid viewports and newpage = FALSE)
# set up grid layout
library(grid)
```

```
grid.newpage()
top_vp <- viewport(layout = grid.layout(nrow = 1, ncol = 2,
  widths = unit(c(.4, .6), units = "npc")))
col1_vp <- viewport(layout.pos.row = 1, layout.pos.col = 1, name = "col1_vp")
col2_vp <- viewport(layout.pos.row = 1, layout.pos.col = 2, name = "col2_vp")
splot <- vpTree(top_vp, vpList(col1_vp, col2_vp))
pushViewport(splot)

seekViewport("col1_vp")
o <- seriate(d)
pimage(x, c(o, NA), col_labels = FALSE, main = "Data",
  newpage = FALSE)

seekViewport("col2_vp")
## add the reordered dissimilarity matrix for rows
pimage(d, o, main = "Distances",
  newpage = FALSE)

popViewport(0)

##------------------------------------------------------------
## ggplot2 Examples
if (require("ggplot2")) {

library("ggplot2")

set.seed(1234)
data(iris)
x <- as.matrix(iris[sample(nrow(iris), 20) , -5])

ggpimage(x)

# Show all labels and flip axes, reverse columns
ggpimage(x, prop = TRUE)
ggpimage(x, flip_axes = TRUE)
ggpimage(x, reverse_columns = TRUE)


# A matrix with positive and negative values
x_scaled <- scale(x)
ggpimage(x_scaled)

# Use reordering
ggpimage(x_scaled, order = TRUE)
ggpimage(x_scaled, order = "Heatmap")

## Example: Distance Matrix
# Show a reordered distance matrix (distances between rows).
# Dark means low distance. The aspect ratio is automatically fixed to 1:1
# using prop = TRUE
d <- dist(x)
ggpimage(d)
ggpimage(d, order = TRUE)
```

```
# Suppress the upper triangle and diagonal
ggpimage(d, order = TRUE, upper_tri = FALSE, diag = FALSE)

# Show only distances that are smaller than 2 using limits on z.
ggpimage(d, order = TRUE, zlim = c(0, 2))

## Example: Correlation Matrix
# we calculate correlation between rows and seriate the matrix
r <- cor(t(x))
o <- seriate(as.dist(sqrt(1 - r)))
ggpimage(r, order = c(o, o),
  upper_tri = FALSE, diag = FALSE,
  zlim = c(-1, 1),
  reverse_columns = TRUE) + labs(title = "Correlation matrix")

## Example: Custom themes and colors
# Reorder matrix, use custom colors, add a title,
# and hide colorkey.
ggpimage(x) +
  theme(legend.position = "none") +
  labs(title = "Random Data") + xlab("Variables")

# Add lines
ggpimage(x) +
  geom_hline(yintercept = seq(0, nrow(x)) + .5) +
  geom_vline(xintercept = seq(0, ncol(x)) + .5)

# Use ggplot2 themes with theme_set
old_theme <- theme_set(theme_linedraw())
ggpimage(d)
theme_set(old_theme)

# Use custom color palettes: Gray scale, Colorbrewer (provided in ggplot2) and colorspace
ggpimage(d, order = seriate(d), upper_tri = FALSE) +
  scale_fill_gradient(low = "black", high = "white", na.value = "white")

ggpimage(d, order = seriate(d), upper_tri = FALSE) +
  scale_fill_distiller(palette = "Spectral", direction = +1, na.value = "white")

ggpimage(d, order = seriate(d), upper_tri = FALSE) +
  colorspace::scale_fill_continuous_sequential("Reds", rev = FALSE, na.value = "white")
}
```

Psych24 *Results of 24 Psychological Test for 8th Grade Students*

Description

A data set collected by Holzinger and Swineford (1939) which consists of the results of 24 psychological tests given to 145 seventh and eighth grade students in a Chicago suburb. This data set

contains the correlation matrix for the 24 test results. The data set was also used as an example for visualization of cluster analysis by Ling (1973).

Format

A 24 x 24 correlation matrix.

References

Holzinger, K. L., Swineford, F. (1939): A study in factor analysis: The stability of a bi-factor solution. *Supplementary Educational Monograph,* No. **48**. Chicago: University of Chicago Press.

Ling, R. L. (1973): A computer generated aid for cluster analysis. *Communications of the ACM,* **16**(6), pp. 355–361.

See Also

Other data: `Chameleon`, `Irish`, `Munsingen`, `SupremeCourt`, `Townships`, `Wood`, `Zoo`, `create_lines_data()`, `is.robinson()`

Examples

```
data("Psych24")

## create a dist object and also get rid of the one negative entry in the
## correlation matrix
d <- as.dist(1 - abs(Psych24))

pimage(d)

## do hclust as in Ling (1973)
hc <- hclust(d, method = "complete")
plot(hc)

pimage(d, hc)

## use seriation
order <- seriate(d, method = "tsp")
#order <- seriate(d, method = "tsp", control = list(method = "concorde"))
pimage(d, order)
```

`register_DendSer` *Register Seriation Methods from Package DendSer*

Description

Register the DendSer dendrogram seriation method and the ARc criterion (Earle and Hurley, 2015) for use with `seriate()`.

Usage

```
register_DendSer()
```

Details

Registers the method "DendSer" for seriate. DendSer is a fast heuristic for reordering dendrograms developed by Earle and Hurley (2015) able to use different criteria.

control for seriate() with method "DendSer" accepts the following parameters:

- "h" or "method": A dendrogram or a method for hierarchical clustering (see hclust). Default: complete-link.
- "criterion": A seriation criterion to optimize (see list_criterion_methods("dist"). Default: "BAR" (Banded anti-Robinson from with 20% band width).
- "verbose": a logical; print progress information?
- "DendSer_args": additional arguments for DendSer::DendSer().

For convenience, the following methods (for different cost functions) are also provided:

- "DendSer_ARc" (anti-robinson form),
- "DendSer_BAR" (banded anti-Robinson form),
- "DendSer_LPL" (lazy path length),
- "DendSer_PL" (path length).

Note: Package **DendSer** needs to be installed.

Value

Nothing.

Author(s)

Michael Hahsler based on code by Catherine B. Hurley and Denise Earle

References

D. Earle, C. B. Hurley (2015): Advances in dendrogram seriation for application to visualization. *Journal of Computational and Graphical Statistics,* **24**(1), 1–25.

See Also

DendSer::DendSer()

Other seriation: register_GA(), register_optics(), register_smacof(), register_tsne(), register_umap(), register_vegan(), registry_for_seriation_methods, seriate(), seriate_best()

Examples

```
## Not run:
register_DendSer()
get_seriation_method("dist", "DendSer")

d <- dist(random.robinson(20, pre=TRUE))

## use Banded AR form with default clustering (complete-link)
o <- seriate(d, "DendSer_BAR")
pimage(d, o)

## use different hclust method (Ward) and AR as the cost function for
## dendrogram reordering
o <- seriate(d, "DendSer", control = list(method = "ward.D2", criterion = "AR"))
pimage(d, o)

## End(Not run)
```

---

register_GA *Register a Genetic Algorithm Seriation Method*

Description

Register a GA-based seriation metaheuristic for use with `seriate()`.

Usage

```
register_GA()

gaperm_mixedMutation(ismProb = 0.8)
```

Arguments

ismProb	probability to use `GA::gaperm_ismMutation()` (inversion) versus `GA::gaperm_simMutation()` (simple insertion).

Details

Registers the method `"GA"` for `seriate()`. This method can be used to optimize any criterion in package **seriation**.

The GA uses by default the ordered cross-over (OX) operator. For mutation, the GA uses a mixture of simple insertion and simple inversion operators. This mixed operator is created using `seriation::gaperm_mixedMutation(ismProb = .8)`, where `ismProb` is the probability that the simple insertion mutation operator is used. See package **GA** for a description of other available cross-over and mutation operators for permutations. The appropriate operator functions in **GA** start with `gaperm_`.

We warm start the GA using "suggestions" given by several heuristics. Set "suggestions" to NA to start with a purely random initial population.

See Example section for available control parameters.

Note: Package **GA** needs to be installed.

Value

Nothing.

Author(s)

Michael Hahsler

References

Luca Scrucca (2013): GA: A Package for Genetic Algorithms in R. *Journal of Statistical Software,* **53**(4), 1–37. URL doi:10.18637/jss.v053.i04.

See Also

Other seriation: register_DendSer(), register_optics(), register_smacof(), register_tsne(), register_umap(), register_vegan(), registry_for_seriation_methods, seriate(), seriate_best()

Examples

```
## Not run:
register_GA()
get_seriation_method("dist", "GA")

data(SupremeCourt)
d <- as.dist(SupremeCourt)

## optimize for linear seriation criterion (LS)
o <- seriate(d, "GA", criterion = "LS", verbose = TRUE)
pimage(d, o)

## Note that by default the algorithm is already seeded with a LS heuristic.
## This run is no warm start (no suggestions) and increase run to 100
o <- seriate(d, "GA", criterion = "LS", suggestions = NA, run = 100,
  verbose = TRUE)
pimage(d, o)

o <- seriate(d, "GA", criterion = "LS", suggestions = NA, run = 100,
  verbose = TRUE,  )

pimage(d, o)

## End(Not run)
```

`register_optics` *Register Seriation Based on OPTICS*

Description

Use ordering points to identify the clustering structure (OPTICS) for `seriate()`.

Usage

```
register_optics()
```

Details

Registers the method `"optics"` for `seriate()`. This method applies the OPTICS ordering algorithm implemented in `dbscan::optics()` to create an ordering.

Note: Package **dbscan** needs to be installed.

Value

Nothing.

References

Mihael Ankerst, Markus M. Breunig, Hans-Peter Kriegel, Joerg Sander (1999). OPTICS: Ordering Points To Identify the Clustering Structure. *ACM SIGMOD international conference on Management of data,* ACM Press, pp. 49-60. doi:10.1145/304181.304187

See Also

`dbscan::optics()`.

Other seriation: `register_DendSer()`, `register_GA()`, `register_smacof()`, `register_tsne()`, `register_umap()`, `register_vegan()`, `registry_for_seriation_methods`, `seriate()`, `seriate_best()`

Examples

```
## Not run:
register_optics()
get_seriation_method("dist", "optics")

d <- dist(random.robinson(50, pre=TRUE, noise=.1))

o <- seriate(d, method = "optics")
pimage(d, o)

## End(Not run)
```

`register_smacof` *Register Seriation Methods from Package smacof*

Description

Registers the `"MDS_smacof"` method for `seriate()` based on multidimensional scaling using stress majorization and the corresponding `"smacof_stress0"` criterion implemented in package smacof (de Leeuw & Mair, 2009).

Usage

```
register_smacof()
```

Details

Seriation method `"smacof"` implements stress majorization with several transformation functions. These functions are passed on as the type control parameter. We default to `"ratio"`, which together with `"interval"` performs metric MDS. `"ordinal"` can be used for non-metric MDS. See `smacof::smacofSym()` for details on the control parameters.

The corresponding criterion called `"smacof_stress0"` is also registered. There additional parameter `type` is used to specify the used transformation function. It should agree with the function used for seriation. See `smacof::stress0()` for details on the stress calculation.

Note: Package **smacof** needs to be installed.

Value

Nothing.

References

Jan de Leeuw, Patrick Mair (2009). Multidimensional Scaling Using Majorization: SMACOF in R. *Journal of Statistical Software, 31(3),* 1-30. doi:10.18637/jss.v031.i03

See Also

Other seriation: `register_DendSer()`, `register_GA()`, `register_optics()`, `register_tsne()`, `register_umap()`, `register_vegan()`, `registry_for_seriation_methods`, `seriate()`, `seriate_best()`

Examples

```
## Not run:
register_smacof()

get_seriation_method("dist", "MDS_smacof")

d <- dist(random.robinson(20, pre = TRUE))

## use Banded AR form with default clustering (complete-link)
```

```
o <- seriate(d, "MDS_smacof", verbose = TRUE)
pimage(d, o)

# recalculate stress for the order
MDS_stress(d, o)

# ordinal MDS. stress needs to be calculated using the correct type with stress0
o <- seriate(d, "MDS_smacof", type = "ordinal", verbose = TRUE)
criterion(d, o, method = "smacof_stress0", type = "ordinal")

## End(Not run)
```

`register_tsne` *Register Seriation Based on 1D t-SNE*

Description

Use t-distributed stochastic neighbor embedding (t-SNE) for `seriate()`.

Usage

```
register_tsne()
```

Details

Registers the method `"tsne"` for `seriate()`. This method applies 1D t-SNE to a data matrix or a distance matrix and extracts the order from the 1D embedding. To speed up the process, an initial embedding is created using 1D multi-dimensional scaling (MDS) or principal components analysis (PCA) which is improved by t-SNE.

The `control` parameter `"mds"` or `"pca"` controls if MDS (for distances) or PCA (for data matrices) is used to create an initial embedding. See `Rtsne::Rtsne()` to learn about the other available `control` parameters.

Perplexity is automatically set as the minimum between 30 and the number of observations. It can be also specified using the control parameter `"perplexity"`.

Note: Package **Rtsne** needs to be installed.

Value

Nothing.

References

van der Maaten, L.J.P. & Hinton, G.E., 2008. Visualizing High-Dimensional Data Using t-SNE. *Journal of Machine Learning Research,* **9**, pp.2579-2605.

See Also

Rtsne::Rtsne()

Other seriation: register_DendSer(), register_GA(), register_optics(), register_smacof(), register_umap(), register_vegan(), registry_for_seriation_methods, seriate(), seriate_best()

Examples

```
## Not run:
register_tsne()

# distances
get_seriation_method("dist", "tsne")

data(SupremeCourt)
d <- as.dist(SupremeCourt)

o <- seriate(d, method = "tsne", verbose = TRUE)
pimage(d, o)

# look at the returned configuration and plot it
attr(o[[1]], "configuration")
plot_config(o)

# the t-SNE results are also available as an attribute (see ? Rtsne::Rtsne)
attr(o[[1]], "model")

## matrix
get_seriation_method("matrix", "tsne")

data("Zoo")
x <- Zoo

x[,"legs"] <- (x[,"legs"] > 0)

# t-SNE does not allow duplicates
x <- x[!duplicated(x), , drop = FALSE]

class <- x$class
label <- rownames(x)
x <- as.matrix(x[,-17])

o <- seriate(x, method = "tsne", eta = 10, verbose = TRUE)
pimage(x, o, prop = FALSE, row_labels = TRUE, col_labels = TRUE)

# look at the row embedding
plot_config(o[[1]], col = class)

## End(Not run)
```

register_umap *Register Seriation Based on 1D UMAP*

Description

Use uniform manifold approximation and projection (UMAP) to embed the data on the number line and create a order for `seriate()`.

Usage

```
register_umap()
```

Details

Registers the method `"umap"` for `seriate()`. This method applies 1D UMAP to a data matrix or a distance matrix and extracts the order from the 1D embedding.

Control parameter `n_epochs` can be increased to find a better embedding.

The returned seriation permutation vector has an attribute named `embedding` containing the umap embedding.

Note: Package **umap** needs to be installed.

Value

Nothing.

References

McInnes, L and Healy, J, UMAP: Uniform Manifold Approximation and Projection for Dimension Reduction, ArXiv e-prints 1802.03426, 2018.

See Also

`umap::umap()` in **umap**.

Other seriation: `register_DendSer()`, `register_GA()`, `register_optics()`, `register_smacof()`, `register_tsne()`, `register_vegan()`, `registry_for_seriation_methods`, `seriate()`, `seriate_best()`

Examples

```
## Not run:
register_umap()

## distances
get_seriation_method("dist", "umap")

data(SupremeCourt)
d <- as.dist(SupremeCourt)
```

```
o <- seriate(d, method = "umap", verbose = TRUE)
pimage(d, o)

# look at the returned embedding and plot it
attr(o[[1]], "configuration")
plot_config(o)

## matrix
get_seriation_method("matrix", "umap")

data("Zoo")
Zoo[,"legs"] <- (Zoo[,"legs"] > 0)
x <- as.matrix(Zoo[,-17])
label <- rownames(Zoo)
class <- Zoo$class

o <- seriate(x, method = "umap", verbose = TRUE)
pimage(x, o)

plot_config(o[[1]], col = class)

## End(Not run)
```

`register_vegan` *Register Seriation Methods from Package vegan*

Description

Register the `"isomap"`, `"monoMDS"`, and `"metaMDS"` seriation methods for dissimilarity matrices. These methods use the corresponding functions from package **vegan**.

Usage

```
register_vegan()
```

Details

Note: Package **vegan** needs to be installed.

Value

Nothing.

See Also

Other seriation: `register_DendSer()`, `register_GA()`, `register_optics()`, `register_smacof()`, `register_tsne()`, `register_umap()`, `registry_for_seriation_methods`, `seriate()`, `seriate_best()`

Examples

```
## Not run:
register_vegan()
list_seriation_methods("dist")

## End(Not run)
```

`registry_for_criterion_methods` *Registry for Criterion Methods*

Description

A registry to manage methods used by `criterion()` to calculate a criterion value given data and a permutation.

Usage

```
registry_criterion

list_criterion_methods(kind, names_only = TRUE)

get_criterion_method(kind, name)

set_criterion_method(
  kind,
  name,
  fun,
  description = NULL,
  merit = NA,
  control = list(),
  verbose = FALSE,
  ...
)

## S3 method for class 'criterion_method'
print(x, ...)
```

Arguments

`kind`	the data type the method works on. For example, `"dist"`, `"matrix"` or `"array"`.
`names_only`	logical; return only the method name. `FALSE` returns also the method descriptions.
`name`	the name for the method used to refer to the method in the function `criterion()`.
`fun`	a function containing the method's code.
`description`	a description of the method. For example, a long name.

merit	logical; indicating if the criterion measure is a merit (TRUE) or a loss (FALSE) measure.
control	a list with control arguments and default values.
verbose	logical; print a message when a new method is registered.
...	further information that is stored for the method in the registry.
x	an object of class "criterion_method" to be printed.

Details

All methods below are convenience methods for the registry named registry_criterion.

list_criterion_method() lists all available methods for a given data type (kind). The result is a vector of character strings with the short names of the methods. If kind is missing, then a list of methods is returned.

get_criterion_method() returns information (including the implementing function) about a given method in form of an object of class "criterion_method".

With set_criterion_method() new criterion methods can be added by the user. The implementing function (fun) needs to have the formal arguments x, order, ..., where x is the data object, order is an object of class ser_permutation_vector and ... can contain additional information for the method passed on from criterion(). The implementation has to return the criterion value as a scalar.

Value

- list_criterion_method() results is a vector of character strings with the names of the methods used for criterion().
- get_criterion_method() returns a given method in form of an object of class "criterion_method".

Author(s)

Michael Hahsler

See Also

This registry uses registry::registry.

Other criterion: criterion()

Examples

```
## the registry
registry_criterion

# List all criterion calculation methods by type
list_criterion_methods()

# List methods for matrix
list_criterion_methods("matrix")

# get more description
```

```
list_criterion_methods("matrix", names_only = FALSE)

# get a specific method
get_criterion_method(kind = "dist", name = "AR_d")

# Define a new method (sum of the diagonal elements)

## 1. implement a function to calculate the measure
criterion_method_matrix_foo <- function(x, order, ...) {
if(!is.null(order)) x <- permute(x,order)
    sum(diag(x))
}

## 2. Register new method
set_criterion_method("matrix", "DiagSum", criterion_method_matrix_foo,
    description = "Calculated the sum of all diagonal entries", merit = FALSE)

list_criterion_methods("matrix")
get_criterion_method("matrix", "DiagSum")

## 3. use all criterion methods (including the new one)
criterion(matrix(1:9, ncol = 3))
```

`registry_for_seriation_methods` *Registry for Seriation Methods*

Description

A registry to manage methods used by `seriate()`.

Usage

```
registry_seriate

list_seriation_methods(kind, names_only = TRUE)

get_seriation_method(kind, name)

set_seriation_method(
  kind,
  name,
  definition,
  description = NULL,
  control = list(),
  randomized = FALSE,
  optimizes = NA_character_,
  verbose = FALSE,
  ...
```

```
)

## S3 method for class 'seriation_method'
print(x, ...)
```

Arguments

kind	the data type the method works on. For example, "dist", "matrix" or "array". If missing, then methods for any type are shown.
names_only	logical; return only the method name. FALSE returns also the method descriptions.
name	the name for the method used to refer to the method in seriate().
definition	a function containing the method's code.
description	a description of the method. For example, a long name.
control	a list with control arguments and default values.
randomized	logical; does the algorithm use randomization and re-running the algorithm several times will lead to different results (see: seriate_rep()).
optimizes	what criterion does the algorithm try to optimize (see: list_criterion_methods()).
verbose	logical; print a message when a new method is registered.
...	further information that is stored for the method in the registry.
x	an object of class "seriation_method" to be printed.

Details

The functions below are convenience function for the registry registry_seriate.

list_seriation_method() lists all available methods for a given data type (kind) (e.g., "dist", "matrix"). The result is a vector of character strings with the method names that can be used in function seriate(). If kind is missing, then a list of methods is returned.

get_seriation_method() returns detailed information for a given method in form of an object of class "seriation_method". The information includes a description, parameters and the implementing function.

With set_seriation_method() new seriation methods can be added by the user. The implementing function (definition) needs to have the formal arguments x, control and, for arrays and matrices margin, where x is the data object and control contains a list with additional information for the method passed on from seriate(), and margin is a vector specifying what dimensions should be seriated. The implementation has to return a list of objects which can be coerced into ser_permutation_vector objects (e.g., integer vectors). The elements in the list have to be in corresponding order to the dimensions of x.

Value

- list_seriation_method() result is a vector of character strings with the names of the methods. These names are used for methods in seriate().
- get_seriation_method() returns a given method in form of an object of class "seriation_method".

Author(s)

Michael Hahsler

See Also

This registry uses registry::registry.

Other seriation: register_DendSer(), register_GA(), register_optics(), register_smacof(), register_tsne(), register_umap(), register_vegan(), seriate(), seriate_best()

Examples

```
# Registry
registry_seriate

# List all seriation methods by type
list_seriation_methods()

# List methods for matrix seriation
list_seriation_methods("matrix")

get_seriation_method(name = "BEA")

# Example for defining a new seriation method (reverse identity function for matrix)

# 1. Create the seriation method: Reverse the row order
#    (NA means no seriation is applied to columns)
seriation_method_reverse_rows <- function(x, control = NULL, margin = c(1, 2)) {
    list(rev(seq(nrow(x))), NA)[margin]
}

# 2. Register new method
set_seriation_method("matrix", "Reverse_rows", seriation_method_reverse_rows,
    description = "Reverse identity order", control = list())

list_seriation_methods("matrix")
get_seriation_method("matrix", "reverse_rows")

# 3. Use the new seriation methods
seriate(matrix(1:12, ncol = 3), "reverse_rows")
```

---

reorder.hclust *Reorder Dendrograms using Optimal Leaf Ordering*

Description

Reorder method for dendrograms for optimal leaf ordering.

Usage

```
## S3 method for class 'hclust'
reorder(x, dist, method = "OLO", ...)
```

Arguments

x	an object of class hclust.
dist	an object of class dist with dissimilarities between the objects in x.
method	a character string with the name of the used measure. Available are: • "OLO" (optimal leaf ordering; Bar-Joseph et al., 2001) implemented in this package and • "GW" (Gruvaeus and Wainer, 1972) from package **gclus**.
...	further arguments are currently ignored.

Details

Minimizes the distance between neighboring objects (leaf nodes) in the dendrogram by flipping the order of subtrees. The algorithm by Gruvaeus and Wainer is implemented in package **gclus** (Hurley 2004).

Value

A reordered hclust object.

Author(s)

Michael Hahsler

References

Bar-Joseph, Z., E. D. Demaine, D. K. Gifford, and T. Jaakkola. (2001): Fast Optimal Leaf Ordering for Hierarchical Clustering. *Bioinformatics,* **17**(1), 22–29.

Gruvaeus, G. and Wainer, H. (1972): Two Additions to Hierarchical Cluster Analysis, *British Journal of Mathematical and Statistical Psychology,* **25**, 200–206.

Hurley, Catherine B. (2004): Clustering Visualizations of Multidimensional Data. *Journal of Computational and Graphical Statistics,* **13**(4), 788–806.

See Also

gclus::reorder.hclust()

Other permutation: get_order(), permutation_vector2matrix(), permute(), ser_dist(), ser_permutation(), ser_permutation_vector()

Examples

```
## cluster European cities by distance
data("eurodist")
d <- as.dist(eurodist)
hc <- hclust(eurodist)

## plot original dendrogram and the reordered dendrograms
plot(hc)
plot(reorder(hc, d, method = "GW"))
plot(reorder(hc, d, method = "OLO"))
```

---

ser_dist *Dissimilarities and Correlations Between Seriation Orders*

Description

Calculates dissimilarities/correlations between seriation orders in a list of type ser_permutation_vector.

Usage

```
ser_dist(x, y = NULL, method = "spearman", reverse = TRUE, ...)

ser_cor(x, y = NULL, method = "spearman", reverse = TRUE, test = FALSE)

ser_align(x, method = "spearman")
```

Arguments

x	set of seriation orders as a list with elements which can be coerced into ser_permutation_vector objects.
y	if not NULL then a single seriation order can be specified. In this case x has to be a single seriation order and not a list.
method	a character string with the name of the used measure. Available measures are for correlation and distances are "kendall","spearman" and "ppc" (positional proximity coefficient). For distances only the additional methods "manhattan", "euclidean", "hamming", and "aprd" (absolute pairwise rank differences) are also available.
reverse	a logical indicating if the revers orders should also be checked in for rank-based methods.
...	Further arguments passed on to the method.
test	a logical indicating if a correlation test should be performed.

Details

For seriation, an order and its reverse are considered identical and are often just an artifact due to the method that creates the order. This is one of the major differences between seriation orders and rankings which impacts how correlations and similarities between seriation orders are calculated. The default setting `reverse = TRUE` corrects for this issue.

`ser_cor()` calculates the correlation between two seriation orders. For ranking-based correlation measures (Spearman and Kendall) the absolute value of the correlation is returned. This effectively corrects for correlations between reversed orders but has the effect that no negative correlations exist. For `test = TRUE`, the appropriate test for association is performed and a matrix with p-values is returned as the attribute `"p-value"`. Note that no correction for multiple testing is performed.

For `ser_dist()`, the correlation coefficients (Kendall's tau and Spearman's rho) are converted into a dissimilarity by taking one minus the correlation value. The Manhattan distance between the ranks in a linear order is equivalent to Spearman's footrule metric (Diaconis 1988). For the non-correlation based measures, `reverse = TRUE` returns the pairwise minima using also the reversed order.

Two precedence invariant measure especially developed for seriation are available. Here `reverse` is ignored.

The positional proximity coefficient (ppc) is a precedence invariant measure based on product of the squared positional distances in two permutations defined as (see Goulermas et al 2016):

$$d_{ppc}(R,S) = 1/h \sum_{j=2}^{n} \sum_{i=1}^{j-1} (\pi_R(i) - \pi_R(j))^2 * (\pi_S(i) - \pi_S(j))^2,$$

where R and S are two seriation orders, pi_R and pi_S are the associated permutation vectors and h is a normalization factor. The associated generalized correlation coefficient is defined as $1 - d_{ppc}$.

The absolute pairwise rank difference (aprd) is also precedence invariant and defined as a distance measure:

$$d_{aprd}(R,S) = \sum_{j=2}^{n} \sum_{i=1}^{j-1} ||\pi_R(i) - \pi_R(j)| - |\pi_S(i) - \pi_S(j)||^p,$$

where p is the power which can be passed on as parameter `p` and is by default set to 2.

`ser_align()` tries to normalize the direction in a list of seriations such that ranking-based methods can be used. We add for each permutation also the reversed order to the set and then use a modified version of Prim's algorithm for finding a minimum spanning tree (MST) to choose if the original seriation order or its reverse should be used. We retain the direction of each order that is added to the MST first. Every time an order is added, its reverse is removed from the possible remaining orders.

Value

- `ser_dist()` returns an object of class stats::dist.
- `ser_align()` returns a new list with elements of class ser_permutation.

Author(s)

Michael Hahsler

References

P. Diaconis (1988): *Group Representations in Probability and Statistics,* Institute of Mathematical Statistics, Hayward, CA.

J.Y. Goulermas, A. Kostopoulos, and T. Mu (2016): A New Measure for Analyzing and Fusing Sequences of Objects. *IEEE Transactions on Pattern Analysis and Machine Intelligence* **38**(5):833-48. doi:10.1109/TPAMI.2015.2470671

See Also

Other permutation: `get_order()`, `permutation_vector2matrix()`, `permute()`, `reorder.hclust()`, `ser_permutation()`, `ser_permutation_vector()`

Examples

```
set.seed(1234)
## seriate dist of 50 flowers from the iris data set
data("iris")
x <- as.matrix(iris[-5])
x <- x[sample(1:nrow(x), 50), ]
rownames(x) <- 1:50
d <- dist(x)

## Create a list of different seriations
methods <- c("HC_complete", "OLO", "GW", "VAT",
  "TSP", "Spectral", "MDS", "Identity", "Random")

os <- sapply(methods, function(m) {
  cat("Doing", m, "... ")
  tm <- system.time(o <- seriate(d, method = m))
  cat("took", tm[3],"s.\n")
  o
})

## Compare the methods using distances. Default is based on
## Spearman's rank correlation coefficient where reverse orders are
## also considered.
ds <- ser_dist(os)
hmap(ds, margins = c(7,7))

## Compare using correlation between orders. Reversed orders have
## negative correlation!
cs <- ser_cor(os, reverse = FALSE)
hmap(cs, margins = c(7,7))

## Compare orders by allowing orders to be reversed.
## Now all but random and identity are highly positive correlated
cs2 <- ser_cor(os, reverse = TRUE)
```

```
hmap(cs2, margins = c(7,7))

## A better approach is to align the direction of the orders first
## and then calculate correlation.
os_aligned <- ser_align(os)
cs3 <- ser_cor(os_aligned, reverse = FALSE)
hmap(cs3, margins = c(7,7))

## Compare the orders using clustering. We use Spearman's foot rule
## (Manhattan distance of ranks). In order to use rank-based method,
## we align the direction of the orders.
os_aligned <- ser_align(os)
ds <- ser_dist(os_aligned, method = "manhattan")
plot(hclust(ds))
```

`ser_permutation` *Class ser_permutation – A Collection of Permutation Vectors for Seriation*

Description

The class `ser_permutation` is a collection of permutation vectors (see class ser_permutation_vector), one for each dimension (mode) of the data to be permuted.

Usage

```
ser_permutation(x, ...)

## S3 method for class 'ser_permutation'
print(x, ...)

## S3 method for class 'ser_permutation'
summary(object, ...)

## S3 method for class 'ser_permutation'
c(..., recursive = FALSE)

## S3 method for class 'ser_permutation'
object[i, ...]
```

Arguments

`x, object`	an object of class `ser_permutation_vector` or any object which can be converted into a object of class `ser_permutation` (e.g. an integer vector).
`...`	vectors for further dimensions.
`recursive`	ignored.
`i`	index of the dimension(s) to extract.

Value

An object of class ser_permutation.

Author(s)

Michael Hahsler

See Also

Other permutation: get_order(), permutation_vector2matrix(), permute(), reorder.hclust(), ser_dist(), ser_permutation_vector()

Examples

```
o <- ser_permutation(1:5, 10:1)
o

## length (number of dimensions)
length(o)

## get permutation vector for 2nd dimension
get_order(o, 2)

## reverse dimensions
o[2:1]

## combine
o <- c(o, ser_permutation(1:15))
o

## get an individual permutation
o[[2]]

## reverse the order of a permutation
o[[2]] <- rev(o[[2]])
get_order(o,2)
```

---

ser_permutation_vector

Class ser_permutation_vector – A Single Permutation Vector for Seriation

---

Description

The class ser_permutation_vector represents a single permutation vector.

Usage

```
ser_permutation_vector(x, method = NULL)

## S3 method for class 'ser_permutation_vector'
c(..., recursive = FALSE)

## S3 method for class 'ser_permutation_vector'
rev(x)

get_method(x, printable = FALSE)

## S3 method for class 'ser_permutation_vector'
length(x)

## S3 method for class 'ser_permutation_vector'
print(x, ...)

## S3 method for class 'ser_permutation_vector'
summary(object, ...)
```

Arguments

x, object	an object if class ser_permutation_vector. Options for the constructor are: (1) an integer permutation vector, (2) an object of class hclust, (3) a numeric vector with a MDS configuration, or (4) NA to indicate a identity permutation.
method	a string representing the method used to obtain the permutation vector.
...	further arguments.
recursive	ignored
printable	a logical; prints "unknown" instead of NULL for non-existing methods.

Details

A permutation vector maps a set of n objects $\{O_1, O_2, ..., O_n\}$ onto itself.

Ordering Representation: In **seriation** we represent a permutation π as a vector which lists the objects' indices in their permuted order. This can be seen as replacing the object in position i with the object in position $\pi(i)$. For example, the permutation vector $\langle 3, 1, 2\rangle$ indicates that in first position is the object with index 3 then the object with index 1 and finally the object with index 2. This representation is often called a (re)arrangement or ordering. The ordering can be extracted from a permutation vector object via get_order(). Such an ordering can be directly used to subset the list of original objects with "[" to apply the permutation.

Rank Representation: An alternative way to specify a permutation is via a list of the ranks of the objects after permutation. This representation is often called a map or substitution. Ranks can be extracted from a permutation vector using get_rank().

Permutation Matrix: Another popular representation is a permutation matrix which performs permutations using matrix multiplication. A permutation matrix can be obtained using get_permutation_matrix().

ser_permutation_vector objects are usually packed into a ser_permutation object which is a collection (a list) of k permutation vectors for k-mode data.

The constructor ser_permutation_vector() checks if the permutation vector is valid (i.e. if all integers occur exactly once).

Value

The constructor ser_permutation_vector() returns an object a ser_permutation_vector

Author(s)

Michael Hahsler

See Also

Other permutation: get_order(), permutation_vector2matrix(), permute(), reorder.hclust(), ser_dist(), ser_permutation()

Examples

```
o <- structure(sample(10), names = paste0("X", 1:10))
o

p <- ser_permutation_vector(o, "random")
p

## some methods
length(p)
get_method(p)
get_order(p)
get_rank(p)
get_permutation_matrix(p)

r <- rev(p)
r
get_order(r)

## create a symbolic identity permutation vector (with unknown length)
## Note: This can be used to permute an object, but methods
##       like length and get_order are not available.
ip <- ser_permutation_vector(NA)
ip
```

---

seriate *Seriate Dissimilarity Matrices, Matrices or Arrays*

Description

Tries to find a linear order for objects using data in the form of a dissimilarity matrix (two-way one-mode data), a data matrix (two-way two-mode data), or a data array (k-way k-mode data). The order can then be used to reorder the dissimilarity matrix/data matrix using `permute()`.

Usage

```
seriate(x, ...)

## S3 method for class 'dist'
seriate(x, method = "Spectral", control = NULL, rep = 1L, ...)

## S3 method for class 'matrix'
seriate(x, method = "PCA", control = NULL, margin = c(1L, 2L), rep = 1L, ...)

## S3 method for class 'array'
seriate(
  x,
  method = "PCA",
  control = NULL,
  margin = seq(length(dim(x))),
  rep = 1L,
  ...
)

## S3 method for class 'data.frame'
seriate(
  x,
  method = "Heatmap",
  control = NULL,
  margin = c(1L, 2L),
  rep = 1L,
  ...
)

## S3 method for class 'table'
seriate(x, method = "CA", control = NULL, margin = c(1L, 2L), ...)
```

Arguments

`x`	the data.
`...`	further arguments are added to the `control` list.
`method`	a character string with the name of the seriation method (default: varies by data type).
`control`	a list of control options passed on to the seriation algorithm.
`rep`	number of random restarts for randomized methods. Uses `seriate_rep()`.

`margin` an integer vector giving the margin indices (dimensions) to be seriated. For example, for a matrix, `1` indicates rows, `2` indicates columns, `c(1 ,2)` means rows and columns. Unseriated margins return the identity seriation order for that margin.

Details

Seriation methods are managed via a registry. See `list_seriation_methods()` for help. In the following, we focus on discussing the methods provided by the package **seriation**. Some methods that use optional packages need to be registered first using a `register_*()` function.

The available control options, default settings, and a description for each algorithm can be retrieved using `get_seriation_method(name = "<seriation method>")`. Some control parameters are also described in more detail below.

Some methods are very slow, and progress can be printed using the control parameter `verbose = TRUE`.

Many seriation methods (heuristically) optimize (minimize or maximize) an objective function often called seriation criterion. The value of the seriation criterion for a given order can be calculated using `criterion()`. In this manual page, we include the criterion, which is optimized by each method using **bold font**. If no criterion is mentioned, then the method does not directly optimize a criterion. A definition of the different seriation criteria can be found on the `criterion()` manual page.

Seriation methods for distance matrices (dist)

One-mode two-way data must be provided as a dist object (not a symmetric matrix). Similarities have to be transformed into dissimilarities. Seriation algorithms fall into different groups based on the approach. In the following, we describe the currently implemented methods. A list with all methods and the available parameters is available here. Hahsler (2017) for a more detailed description and an experimental comparison of the most popular methods.

Dendrogram leaf order

These methods create a dendrogram using hierarchical clustering and then derive the seriation order from the leaf order in the dendrogram. Leaf reordering may be applied.

- **Hierarchical clustering:** `"HC"`, `"HC_single"`, `"HC_complete"`, `"HC_average"`, `"HC_ward"`

  Uses the order of the leaf nodes in a dendrogram obtained by hierarchical clustering as a simple seriation technique. This method applies hierarchical clustering (`stats::hclust()`) to `x`. The clustering method can be given using a `"linkage"` element in the `control` list. If omitted, the default `"complete"` is used. For convenience, the other methods are provided as shortcuts.

- **Reordered by the Gruvaeus and Wainer heuristic:** `"GW"`, `"GW_single"`, `"GW_average"`, `"GW_complete"`, `"GW_ward"` (Gruvaeus and Wainer, 1972)

  Method `"GW"` uses an algorithm developed by Gruvaeus and Wainer (1972) as implemented `gclus::reorder.hclust()` (Hurley 2004). The clusters are ordered at each level so that the objects at the edge of each cluster are adjacent to the nearest object outside the cluster. The method produces a unique order.

  The methods start with a dendrogram created by `hclust()`. As the `"linkage"` element in the `control` list, a clustering method (default `"average"`) can be specified. Alternatively, an stats::hclust object can be supplied using an element named `"hclust"`.

A dendrogram (binary tree) has 2^{n-1} internal nodes (subtrees) and the same number of leaf orderings. That is, at each internal node, the left and right subtree (or leaves) can be swapped or, in terms of a dendrogram, be flipped. The leaf-node reordering to minimize

Minimizes the **Hamiltonian path length (restricted by the dendrogram)**.

- **Reordered by optimal leaf ordering:** `"OLO"`, `"OLO_single"`, `"OLO_average"`, `"OLO_complete"`, `"OLO_ward"` (Bar-Joseph et al., 2001)

  Starts with a dendrogram and produces an optimal leaf ordering that minimizes the sum of the distances along the (Hamiltonian) path connecting the leaves in the given order. The algorithm's time complexity is $O(n^3)$. Note that non-finite distance values are not allowed.

  Minimizes the **Hamiltonian path length (restricted by the dendrogram)**.

- **Dendrogram seriation:** `"DendSer"` (Earle and Hurley, 2015)

  Use heuristic dendrogram seriation to optimize for various criteria. The DendSer code has to be first registered. A detailed description can be found on the manual page for `register_DendSer()`.

Dimensionality reduction

Find a seriation order by reducing the dimensionality to 1 dimension. This is typically done by minimizing a stress measure or the reconstruction error. Note that dimensionality reduction to a single dimension is a very difficult discrete optimization problem. For example, MDS algorithms used for a single dimension tend to end up in local optima (see Maier and De Leeuw, 2015). However, generally, ordering along a single component of MDS provides good results sufficient for applications like visualization.

- **Classical metric multidimensional scaling:** `"MDS"`

  Orders along the 1D classical metric multidimensional scaling. `control` parameters are passed on to `stats::cmdscale()`.

- **Isometric feature mapping:** `"isomap"` (Tenenbaum, 2000)

  Orders along the 1D isometric feature mapping. This method is available after calling `register_vegan()`. `control` parameters are passed on to `vegan::isomap()`

- **Kruskal's non-metric multidimensional scaling:** `"isoMDS"`, `"monoMDS"`, `"metaMDS"` (Kruskal, 1964)

  Orders along the 1D Kruskal's non-metric multidimensional scaling. Package **vegan** provides optional methods. Call `register_vegan()` to register `"monoMDS"` and `"metaMDS"`. It provides an alternative implementation called `monoMDS` and a version that uses random restarts for stability called `metaMDS`. `control` parameters are passed on to `MASS::isoMDS()`, `vegan::monoMDS()` or `vegan::metaMDS()`.

- **Sammon's non-linear mapping:** `"Sammon_mapping"` (Sammon, 1969)

  Orders along the 1D Sammon's non-linear mapping. `control` parameters are passed on to `MASS::sammon()`.

- **Angular order of the first two eigenvectors:** `"MDS_angle"`

  Finds a 2D configuration using MDS (`stats::cmdscale()`) to approximate the eigenvectors of the covariance matrix in the original data matrix. Orders by the angle in this space and splits the order by the largest gap between adjacent angles. A similar method was used by Friendly (2002) to order variables in correlation matrices by angles of first two eigenvectors.

- **Smacof:** `"MDS_smacof"` (de Leeuw and Mair, 2009)

  Perform seriation using stress majorization with several transformation functions. This method has to be registered first using `register_smacof()`.

Optimization

These methods try to optimize a seriation criterion directly, typically using a heuristic approach.

- **Anti-Robinson seriation by simulated annealing:** `"ARSA"` (Brusco et al 2008)

  The algorithm automatically finds a suitable start temperature and calculates the needed number of iterations. The algorithm gets slow for a large number of objects. The speed can be improved by lowering the cooling parameter `"cool"` or increasing the minimum temperature `"tmin"`. However, this will decrease the seriation quality.

  Directly minimizes the **linear seriation criterion (LS).**

- **Complete Enumeration:** `"Enumerate"`

  This method finds the optimal permutation given a seriation criterion by complete enumeration of all permutations. The criterion is specified as the `control` parameters `"criterion"`. Default is the weighted gradient measure. Use `"verbose = TRUE"` to see the progress.

  Note: The number of permutations for n objects is $n!$. Complete enumeration is only possible for tiny problems (<10 objects) and is limited on most systems to a problem size of up to 12 objects.

- **Gradient measure seriation by branch-and-bound:** `"BBURCG"`, `"BBWRCG"` (Brusco and Stahl 2005)

  The method uses branch-and-bound to minimize the **unweighted gradient measure** (`"BBURCG"`) and the **weighted gradient measure** (`"BBWRCG"`). This type of optimization is only feasible for a small number of objects (< 50 objects).

  For BBURCG, the control parameter `"eps"` can be used to relax the problem by defining that a distance needs to be eps larger to count as a violation. This relaxation will improve the speed, but miss some Robinson events. The default value is 0.

- **Genetic Algorithm:** `"GA"`

  The GA code has to be first registered. A detailed description can be found on the manual page for `register_GA()`.

- **Quadratic assignment problem seriation:** `"QAP_LS"`, `"QAP_2SUM"`, `"QAP_BAR"`, `"QAP_Inertia"` (Hahsler, 2017)

  Formulates the seriation problem as a quadratic assignment problem and applies a simulated annealing solver to find a good solution. These methods minimize the **Linear Seriation Problem** (LS) formulation (Hubert and Schultz 1976), the **2-Sum Problem** formulation (Barnard, Pothen, and Simon 1993), the **banded anti-Robinson form** (BAR), or the **inertia criterion**.

  `control` parameters are passed on to `qap::qap()`. An important parameter is `rep` to return the best result from the given number of repetitions with random restarts. The default is 1, but bigger numbers result in better and more stable results.

- **General Simulated Annealing:** `"GSA"` (Hahsler, Hornik, and Buchta, 2023)

  Implement simulated annealing similar to the ARSA method (Brusco et al 2008) which can optimize for any criterion measure defined in **seriation**. By default, the algorithm optimizes for the raw gradient measure, and is warm started with the result of spectral seriation (2-Sum problem) since Hahsler (2017) shows that 2-Sum solutions are similar to solutions for the gradient measure. This method was first introduced in the R package **seriation** version 1.5.0.

  Use `warmstart = "random"` for no warm start.

  The initial temperature `t0` and minimum temperature `tmin` can be set. If `t0` is not set, then it is estimated by sampling uphill moves and setting `t0` such that the median uphill move have

a probability of `tinitialaccept`. Using the cooling rate `cool`, the number of iterations to go for `t0` to `tmin` is calculated.

Several popular local neighborhood functions are provided, and new ones can be defined (see LS). Local moves are tried in each iteration `nlocal` times the number of objects.

Note that this is an R implementation repeatedly calling the criterion function which is very slow.

- **Stochastic greedy local search:** `"SGLS"` (Hahsler, Hornik, and Buchta, 2023)

  Starts with a solution and repeatedly applies a randomly selected local move, accepting only moves that improve the specified criterion. This is stochastic greedy local search over permutations. This method was first introduced in the R package **seriation** version 1.5.0.

  Important `control` parameters:

  - `"criterion"`: the criterion to optimize
  - `"init"`: initial seriation (an order or the name of a seriation method)
  - `"max_iter"`: number of trials

- **Spectral seriation:** `"Spectral"`, `"Spectral_norm"` (Ding and He, 2004)

  Spectral seriation uses a relaxation to minimize the **2-Sum Problem** (Barnard, Pothen, and Simon, 1993). It uses the order of the Fiedler vector of the similarity matrix's (normalized) Laplacian.

  Spectral seriation gives a good trade-off between seriation quality, and scalability (see Hahsler, 2017).

- **Traveling salesperson problem solver:** `"TSP"`

  Uses a traveling salesperson problem solver to minimize the **Hamiltonian path length**. The solvers in **TSP** are used (see `TSP::solve_TSP()`). The solver method can be passed on via the `control` argument, e.g., `control = list(method = "two_opt")`. Default is the est of 10 runs of arbitrary insertion heuristic with 2-opt improvement.

  Since a tour returned by a TSP solver is a connected circle and we are looking for a path representing a linear order, we need to find the best cutting point. Climer and Zhang (2006) suggest adding a dummy city with equal distance to each other city before generating the tour. The place of this dummy city in an optimal tour with minimal length is the best cutting point (it lies between the most distant cities).

Other Methods

- **Identity permutation:** ‘"Identity"
- **Reverse Identity permutation:** ‘"Reverse"
- **Random permutation:** `"Random"`
- **Rank-two ellipse seriation:** `"R2E"` (Chen 2002)

  Rank-two ellipse seriation starts with generating a sequence of correlation matrices $R^1, R^2, \ldots$. R^1 is the correlation matrix of the original distance matrix D (supplied to the function as `x`), and

  $$R^{n+1} = \phi R^n,$$

  where ϕ calculates the correlation matrix.

  The rank of the matrix R^n falls with increasing n. The first R^n in the sequence, which has a rank of 2 is found. Projecting all points in this matrix on the first two eigenvectors, all points fall on an ellipse. The order of the points on this ellipse is the resulting order.

The ellipse can be cut at the two interception points (top or bottom) of the vertical axis with the ellipse. In this implementation, the topmost cutting point is used.

- **Sorting Points Into Neighborhoods:** `"SPIN_STS"`, `"SPIN_NH"` (Tsafrir, 2005)
  Given a weight matrix W, the SPIN algorithms try to minimize the energy for a permutation (matrix P) given by
  $$F(P) = tr(PDP^T W),$$
  where tr denotes the matrix trace.
  `"SPIN_STS"` implements the Side-to-Side algorithm, which tries to push out large distance values. The default weight matrix suggested in the paper with $W = XX^T$ and $X_i = i - (n + 1)/2$ is used. We run the algorithm form `step` (25) iteration and restart the algorithm `nstart` (10) with random initial permutations (default values in parentheses).
  `"SPIN_NH"` implements the neighborhood algorithm (concentrate low distance values around the diagonal) with a Gaussian weight matrix $W_{ij} = exp(-(i-j)^2/n\sigma)$, where n is the size of the dissimilarity matrix and σ is the variance around the diagonal that control the influence of global (large σ) or local (small σ) structure.
  We use the heuristic suggested in the paper for the linear assignment problem. We do not terminate as indicated in the algorithm but run all the iterations since the heuristic does not guarantee that the energy is strictly decreasing. We also implement the heuristic "annealing" scheme where σ is successively reduced. The parameters in `control` are `sigma` which can be a single value or a decreasing sequence (default: 20 to 1 in 10 steps), and `step`, which defines how many update steps are performed before for each value of `alpha`. Via `W_function` a custom function to create W with the function signature `function(n, sigma, verbose)` can be specified.
- **Visual Assessment of (Clustering) Tendency:** `"VAT"` (Bezdek and Hathaway, 2002).
  Creates an order based on Prim's algorithm for finding a minimum spanning tree (MST) in a weighted connected graph representing the distance matrix. The order is given by the order in which the nodes (objects) are added to the MST.

Seriation methods for matrices (matrix)

Two-mode two-way data are general matrices. Some methods also require that the matrix is positive. Data frames and contingency tables (base::table) are converted into a matrix. However, the default methods are different.

Some methods find the row and column order simultaneously, while others calculate them independently. Currently, the following methods are implemented for `matrix`:

Seriating rows and columns simultaneously

Row and column order influence each other.

- **Bond Energy Algorithm:** `"BEA"` (McCormick, 1972).
  The algorithm tries to maximize a non-negative matrix's **Measure of Effectiveness.** Due to the definition of this measure, the tasks of ordering rows and columns are separable and can be solved independently.
  BEA consists of the following three steps:
  1. Place one randomly chosen column.
  2. Try to place each remaining column at each possible position left, right and between the already placed columns and calculate every time the increase in ME. Choose the column and position which gives the largest increase in ME and place the column. Repeat till all columns are placed.

3. Repeat procedure with rows.

The overall procedure amounts to two approximate traveling salesperson problems (TSP) where the distance is the -1 times the ME increase. The BEA algorithm is equivalent to a simple suboptimal TSP heuristic called 'cheapest insertion'. Several consecutive runs of the algorithm might improve the energy if a better initial column/row is chosen.

Arabie and Hubert (1990) question its use with non-binary data if the objective is to find a seriation or one-dimensional orderings of rows and columns.

- **TSP to optimize the Measure of Effectiveness**: `"BEA_TSP"` (Lenstra 1974).
  Since BEA is equivalent to a simple TSP heuristic, we can use better TSP solvers to get better results. Distances between rows are calculated for a $M \times N$ data matrix as

$$d_{jk} = -\sum_{i=1}^{i=M} x_{ij}x_{ik} \ (j,k = 0,1,...,N).$$

  Distances between columns are calculated the same way from the transposed data matrix.

  Solving the two TSP using these distances optimizes the measure of effectiveness. With an exact TSP solver, the optimal solution can be found.

  `control` parameter:

  - `"method"`: a TSP solver method (see `TSP::solve_TSP()`).

- **Unconstrained Brower and Kyle seriation**: `"BK_unconstrained"` (Brower and Kyle 1988).
  Reorders 0-1 matrices to create a block structure along the diagonal. It iteratively reorders by the mean row indices of 1s and mean column indices of 1s till the orders become stable.

  `control` parameter: None

- **Correspondence analysis** `"CA"` (Friendly, 2023)
  This function is designed to help simplify a mosaic plot or other displays of a matrix of frequencies. It calculates a correspondence analysis of the matrix and an order for rows and columns according to the scores on a correspondence analysis dimension.

  This is the default method for contingency tables.

  `control` parameters:

  - `"dim"`: CA dimension used for reordering.
  - `"ca_param"`: List with parameters for the call to `ca::ca()`.

Seriating rows and columns separately using dissimilarities

- **Heatmap seriation:** `"Heatmap"`
  Calculates distances between rows and between columns and then applies seriation so each. This is the default method for data frames.

  `control` parameter:

  - `"seriation_method"`: a list with row and column seriation methods. The special method `"HC_Mean"` is available to use hierarchical clustering with reordering the leaves by the row/column means (see `stats::heatmap()`). Defaults to optimal leaf ordering `"OLO"`.
  - `"seriation_control"`: a list with control parameters for row and column seriation methods.
  - `"dist_fun"`: specify the distance calculation as a function.
  - `"scale"`: `"none"`, `"row"`, or `"col"`.

Seriate rows using the data matrix

These methods need access to the data matrix instead of dissimilarities to reorder objects (rows). Columns can also be reordered by applying the same technique to the transposed data matrix.

- **Order along the 1D locally linear embedding:** `"LLE"`

Performs 1D the non-linear dimensionality reduction method locally linear embedding (see `lle()`).

- **Order along the first principal component:** `"PCA"`

  Uses the projection of the data on its first principal component (using `stats::princomp()`) to determine the order of rows. Performs the same procedure on the transposed matrix to obtain the column order.

  Note that for a distance matrix calculated from `x` with Euclidean distance, this method minimizes the least square criterion.

- **Angular order of the first two PCA components:** `"PCA_angle"`

  For rows, projects the data on the first two principal components and then orders by the angle in this space. The order is split by the largest gap between adjacent angles. A similar method was suggested by Friendly (2002) to order variables in correlation matrices by angles of first two eigenvectors. PCA also computes the eigenvectors of the covariance matrix of the data.

  Performs the same process on the transposed matrix for the column order.

Other methods

- **Angular order of the first two eigenvectors:** `"AOE"` (Friendly 2002)

  This method reordered correlation matrices by the angle in the space spanned by the two largest eigenvectors of the matrix. The order is split by the largest angle gap. This is the original method proposed by Friendly (2002).

- **By row/column mean:** `"Mean"`

  A transformation can be applied before calculating the means. The function is specified as control parameter `"transformation"`. Any function that takes as an input a matrix and returns the transformed matrix can be used. Examples are `scale` or `\(x) x^.5`.

- **Identity permutation:** `"Identity"`
- **Reverse Identity permutation:** `"Reverse"`
- **Random permutation:** `"Random"`

For **general arrays** no built-in methods are currently available.

Value

Returns an object of class ser_permutation.

Author(s)

Michael Hahsler

References

Arabie, P. and L.J. Hubert (1990): The bond energy algorithm revisited, *IEEE Transactions on Systems, Man, and Cybernetics,* **20**(1), 268–274. doi:10.1109/21.47829

Bar-Joseph, Z., E. D. Demaine, D. K. Gifford, and T. Jaakkola. (2001): Fast Optimal Leaf Ordering for Hierarchical Clustering. *Bioinformatics,* **17**(1), 22–29. doi:10.1093/bioinformatics/17.suppl_1.S22

Barnard, S. T., A. Pothen, and H. D. Simon (1993): A Spectral Algorithm for Envelope Reduction of Sparse Matrices. *In Proceedings of the 1993 ACM/IEEE Conference on Supercomputing,* 493–502. Supercomputing '93. New York, NY, USA: ACM. doi:10.1145/169627.169790

Bezdek, J.C. and Hathaway, R.J. (2002): VAT: a tool for visual assessment of (cluster) tendency. *Proceedings of the 2002 International Joint Conference on Neural Networks (IJCNN '02),* Volume: 3, 2225–2230. doi:10.1109/IJCNN.2002.1007487

Brower, J.C. and Kile, K.M. (1988): Sedation of an original data matrix as applied to paleoecology. *Lethaia,* **21**, 79–93. doi:10.1111/j.15023931.1988.tb01756.x

Brusco, M., Koehn, H.F., and Stahl, S. (2008): Heuristic Implementation of Dynamic Programming for Matrix Permutation Problems in Combinatorial Data Analysis. *Psychometrika,* **73**(3), 503–522. doi:10.1007/s1133600790495

Brusco, M., and Stahl, S. (2005): *Branch-and-Bound Applications in Combinatorial Data Analysis.* New York: Springer. doi:10.1007/0387288104

Chen, C. H. (2002): Generalized Association Plots: Information Visualization via Iteratively Generated Correlation Matrices. *Statistica Sinica,* **12**(1), 7–29.

Ding, C. and Xiaofeng He (2004): Linearized cluster assignment via spectral ordering. *Proceedings of the Twenty-first International Conference on Machine learning (ICML '04).* doi:10.1145/1015330.1015407

Climer, S. and Xiongnu Zhang (2006): Rearrangement Clustering: Pitfalls, Remedies, and Applications, *Journal of Machine Learning Research,* **7**(Jun), 919–943.

D. Earle, C. B. Hurley (2015): Advances in dendrogram seriation for application to visualization. *Journal of Computational and Graphical Statistics,* **24**(1), 1–25.

Friendly, M. (2002): Corrgrams: Exploratory Displays for Correlation Matrices. *The American Statistician,* **56**(4), 316–324. doi:10.1198/000313002533

Friendly, M. (2023). *vcdExtra: 'vcd' Extensions and Additions*. R package version 0.8-5, https://CRAN.R-project.org/package=vcdExtra.

Gruvaeus, G. and Wainer, H. (1972): Two Additions to Hierarchical Cluster Analysis, *British Journal of Mathematical and Statistical Psychology,* **25**, 200–206. doi:10.1111/j.20448317.1972.tb00491.x

Hahsler, M. (2017): An experimental comparison of seriation methods for one-mode two-way data. *European Journal of Operational Research,* **257**, 133–143. doi:10.1016/j.ejor.2016.08.066

Hahsler M, Buchta C, Hornik K (2023). *seriation: Infrastructure for Ordering Objects Using Seriation*. R package version 1.5.0/ doi:10.32614/CRAN.package.seriation

Hubert, Lawrence, and James Schultz (1976): Quadratic Assignment as a General Data Analysis Strategy. *British Journal of Mathematical and Statistical Psychology,* **29**(2). Blackwell Publishing Ltd. 190–241. doi:10.1111/j.20448317.1976.tb00714.x

Hurley, Catherine B. (2004): Clustering Visualizations of Multidimensional Data. *Journal of Computational and Graphical Statistics,* **13**(4), 788–806. doi:10.1198/106186004X12425

Kruskal, J.B. (1964). Nonmetric multidimensional scaling: a numerical method. *Psychometrika,* **29**, 115–129.

Lenstra, J.K (1974): Clustering a Data Array and the Traveling-Salesman Problem, *Operations Research,* **22**(2) 413–414. doi:10.1287/opre.22.2.413

Mair P., De Leeuw J. (2015). Unidimensional scaling. In *Wiley StatsRef: Statistics Reference Online,* Wiley, New York. doi:10.1002/9781118445112.stat06462.pub2

McCormick, W.T., P.J. Schweitzer and T.W. White (1972): Problem decomposition and data reorganization by a clustering technique, *Operations Research,* **20**(5), 993–1009. doi:10.1287/opre.20.5.993

Tenenbaum, J.B., de Silva, V. & Langford, J.C. (2000) A global network framework for nonlinear dimensionality reduction. *Science* **290**, 2319-2323.

Tsafrir, D., Tsafrir, I., Ein-Dor, L., Zuk, O., Notterman, D.A. and Domany, E. (2005): Sorting points into neighborhoods (SPIN): data analysis and visualization by ordering distance matrices, *Bioinformatics,* **21**(10) 2301–8. doi:10.1093/bioinformatics/bti329

Sammon, J. W. (1969) A non-linear mapping for data structure analysis. *IEEE Trans. Comput.*, **C-18** 401–409.

See Also

Other seriation: `register_DendSer()`, `register_GA()`, `register_optics()`, `register_smacof()`, `register_tsne()`, `register_umap()`, `register_vegan()`, `registry_for_seriation_methods`, `seriate_best()`

Examples

```
# Show available seriation methods (for dist and matrix)
list_seriation_methods()

# show the description for ARSA
get_seriation_method("dist", name = "ARSA")

### Seriate as distance matrix (for 50 flowers from the iris dataset)
data("iris")
x <- as.matrix(iris[-5])
x <- x[sample(nrow(x), size = 50), ]
d <- dist(x)

order <- seriate(d)
order

pimage(d, main = "Distances (Random Order)")
pimage(d, order, main = "Distances (Reordered)")

# Compare seriation quality
rbind(
        random = criterion(d),
        reordered = criterion(d, order)
    )

# Reorder the distance matrix
```

```
d_reordered <-  permute(d, order)
pimage(d_reordered, main = "Distances (Reordered)")


### Seriate a matrix (50 flowers from iris)

# To make the variables comparable, we scale the data
x <- scale(x, center = FALSE)

# The iris flowers are ordered by species in the data set
pimage(x, main = "original data", prop = FALSE)
criterion(x)

# Apply some methods
order <- seriate(x, method = "BEA_TSP")
pimage(x, order, main = "TSP to optimize ME", prop = FALSE)
criterion(x, order)

order <- seriate(x, method = "PCA")
pimage(x, order, main = "First principal component", prop = FALSE)
criterion(x, order)

order <- seriate(x, method = "heatmap")
pimage(x, order, main = "Heatmap seriation", prop = FALSE)
criterion(x, order)

# reorder the matrix
x_reordered <- permute(x, order)

# create a heatmap seriation manually by calculating
# distances between rows and between columns
order <- c(
    seriate(dist(x), method = "OLO"),
    seriate(dist(t(x)), method = "OLO")
)
pimage(x, order, main = "Heatmap seriation", prop = FALSE)
criterion(x, order)

### Seriate a correlation matrix
corr <- cor(x)

# plot in original order
pimage(corr, main = "Correlation matrix")

# reorder the correlation matrix using the angle of eigenvectors
pimage(corr, order = "AOE", main = "Correlation matrix (AOE)")

# we can also define a distance (we used d = sqrt(1 - r)) and
# then reorder the matrix (rows and columns) using any seriation method.
d <- as.dist(sqrt(1 - corr))
o <- seriate(d, method = "R2E")
corr_reordered <- permute(corr, order = c(o, o))
pimage(corr_reordered, main = "Correlation matrix (R2E)")
```

`seriate_best` *Best Seriation*

Description

Often the best seriation method for a particular dataset is not know and heuristics may produce unstable results. `seriate_best()` and `seriate_rep()` automatically try different seriation methods or rerun randomized methods several times to find the best and order given a criterion measure. `seriate_improve()` uses a local improvement strategy to improve an existing solution.

Usage

```
seriate_best(
  x,
  methods = NULL,
  control = NULL,
  criterion = NULL,
  rep = 10L,
  parallel = TRUE,
  verbose = TRUE,
  ...
)

seriate_rep(
  x,
  method = NULL,
  control = NULL,
  criterion = NULL,
  rep = 10L,
  parallel = TRUE,
  verbose = TRUE,
  ...
)

seriate_improve(
  x,
  order,
  criterion = NULL,
  control = NULL,
  verbose = TRUE,
  ...
)
```

Arguments

`x`	the data.
`methods`	a vector of character string with the name of the seriation methods to try.

control	a list of control options passed on to seriate(). For seriate_best() control needs to be a named list of control lists with the names matching the seriation methods.
criterion	seriate_rep() chooses the criterion specified for the method in the registry. A character string with the criterion to optimize can be specified.
rep	number of times to repeat the randomized seriation algorithm.
parallel	logical; perform replications in parallel. Uses foreach::foreach() if a %dopar% backend (e.g., doParallel::doParallel) is registered.
verbose	logical; show progress and results for different methods
...	further arguments are passed on to the seriate().
method	a character string with the name of the seriation method (default: varies by data type).
order	a ser_permutation object for x or the name of a seriation method to start with.

Details

seriate_rep() rerun a randomized seriation methods to find the best solution given the criterion specified for the method in the registry. A specific criterion can also be specified. Non-stochastic methods are automatically only run once.

seriate_best() runs a set of methods and returns the best result given a criterion. Stochastic methods are automatically randomly restarted several times.

seriate_improve() improves a seriation order using simulated annealing using a specified criterion measure. It uses seriate() with method "GSA", a reduced probability to accept bad moves, and a lower minimum temperature. Control parameters for this method are accepted.

Criterion

If no criterion is specified, then the criterion specified for the method in the registry (see [get_seriation_method()]) is used. For methods with no criterion in the registry (marked as "other"), a default method is used. The defaults are:

- dist: "AR_deviations" - the study in Hahsler (2007) has shown that this criterion has high similarity with most other criteria.
- matrix: "Moore_stress"

Parallel Execution

Some methods support for parallel execution is provided using the foreach package. To use parallel execution, a suitable backend needs to be registered (see the Examples section for using the doParallel backend).

Value

Returns an object of class ser_permutation.

Author(s)

Michael Hahsler

References

Hahsler, M. (2017): An experimental comparison of seriation methods for one-mode two-way data. *European Journal of Operational Research,* **257**, 133–143. doi:10.1016/j.ejor.2016.08.066

See Also

Other seriation: register_DendSer(), register_GA(), register_optics(), register_smacof(), register_tsne(), register_umap(), register_vegan(), registry_for_seriation_methods, seriate()

Examples

```
data(SupremeCourt)
d_supreme <- as.dist(SupremeCourt)

# find best seriation order (tries by by default several fast methods)
o <- seriate_best(d_supreme, criterion = "AR_events", rep = 5)
o
pimage(d_supreme, o)

# run a randomized algorithms several times. It automatically chooses the
# LS criterion. Repetition information is returned as attributes
o <- seriate_rep(d_supreme, "QAP_LS", rep = 5)

attr(o, "criterion")
hist(attr(o, "criterion_distribution"))
pimage(d_supreme, o)

## Not run:
# Using parallel execution on a larger dataset
data(iris)
m_iris <- as.matrix(iris[sample(seq(nrow(iris))),-5])
d_iris <- dist(m_iris)

library(doParallel)
registerDoParallel(cores = detectCores() - 1L)

# seriate rows of the iris data set
o <- seriate_best(d_iris, criterion = "LS")
o

pimage(d_iris, o)

# improve the order to minimize RGAR instead of LS
o_improved <- seriate_improve(d_iris, o, criterion = "RGAR")
pimage(d_iris, o_improved)

# available control parameters for seriate_improve()
get_seriation_method(name = "GSA")

## End(Not run)
```

SupremeCourt *Voting Patterns in the Second Rehnquist U.S. Supreme Court*

Description

Contains a (a subset of the) decisions for the stable 8-yr period 1995-2002 of the second Rehnquist Supreme Court. Decisions are aggregated to the joint probability for disagreement between judges.

Format

A square, symmetric 9-by-9 matrix with the joint probability for disagreement.

Author(s)

Michael Hahsler

References

Sirovich, L. (2003). A pattern analysis of the second Rehnquist
U.S. Supreme Court. _Proceedings of the National Academy of Sciences of the United
States of America,_ **100**, 7432-7437. \doi{10.1073/pnas.1132164100}

See Also

Other data: Chameleon, Irish, Munsingen, Psych24, Townships, Wood, Zoo, create_lines_data(), is.robinson()

Examples

```
data("SupremeCourt")

# a matrix with joint probability of disagreement
SupremeCourt

# show judges in original alphabetical order
d <- as.dist(SupremeCourt)
pimage(d, diag = TRUE, upper_tri = TRUE)

# reorder judges using seriation based on similar decisions
o <- seriate(d)
o

pimage(d, o, diag = TRUE, upper_tri = TRUE)

# Use optimal leaf ordering (hierarchical clustering with reordering)
# which uses a dendrogram
o <- seriate(d, method = "OLO")
o
```

```
plot(o[[1]])

# Use multi-dimensional scaling and show the configuration
o <- seriate(d, method = "sammon")
o

pimage(d, o, diag = TRUE, upper_tri = TRUE)
plot_config(o[[1]])
```

Townships *Bertin's Characteristics of Townships*

Description

This data contains nine characteristics for 16 townships. The data set was used by Bertin (1981) to illustrate that the conciseness of presentation can be improved by seriating the rows and columns.

Format

A matrix with 16 0-1 variables (columns) indicating the presence (1) or absence (0) of characteristics of townships (rows).

Author(s)

Michael Hahsler

References

Bertin, J. (1981): *Graphics and Graphic Information Processing*. Berlin, Walter de Gruyter.

See Also

Other data: Chameleon, Irish, Munsingen, Psych24, SupremeCourt, Wood, Zoo, create_lines_data(), is.robinson()

Examples

```
data("Townships")

## original data
pimage(Townships)
criterion(Townships)

## seriated data using an improved Bond-Energy Algorithm
order <- seriate(Townships, method = "BEA_TSP")
pimage(Townships, order)
criterion(Townships, order)
```

uniscale *Fit an Unidimensional Scaling for a Seriation Order*

Description

Fits an (approximate) unidimensional scaling configuration given an order.

Usage

```
uniscale(d, order, accept_reorder = FALSE, warn = TRUE, ...)

MDS_stress(d, order, refit = TRUE, warn = FALSE)

get_config(x, dim = 1L, ...)

plot_config(x, main, pch = 19, labels = TRUE, pos = 1, cex = 1, ...)
```

Arguments

d	a dissimilarity matrix.
order	a precomputed permutation (configuration) order.
accept_reorder	logical; accept a configuration that does not preserve the requested order. If FALSE, the initial configuration stored in order or, an equally spaced configuration is returned.
warn	logical; produce a warning if the 1D MDS fit does not preserve the given order.
...	additional arguments are passed on to the seriation method.
refit	logical; forces to refit a minimum-stress MDS configuration, even if order contains a configuration.
x	a scaling returned by uniscale() or a ser_permutation with a configuration attribute.
dim	The dimension if x is a ser_permutation object.
main	main plot label
pch	print character
labels	add the object names to the plot
pos	label position for 2D plot (see text()).
cex	label expansion factor.

Details

This implementation uses the method describes in Maier and De Leeuw (2015) to calculate the minimum stress configuration for a given (seriation) order by performing a 1D MDS fit. If the 1D MDS fit does not preserve the given order perfectly, then a warning is produced indicating for how many positions order could not be preserved. The seriation method which is consistent to uniscale is "MDS_smacof" which needs to be registered with register_smacof().

The code is similar to `smacof::uniscale()` (de Leeuw, 2090), but scales to larger datasets since it only uses the permutation given by `order`.

`MDS_stress()` calculates the normalized stress of a configuration given by a seriation order. If the order does not contain a configuration, then a minimum-stress configuration if calculates for the given order.

All distances are first normalized to an average distance of close to 1 using $d_{ij} \frac{\sqrt{n(n-1)/2}}{\sqrt{\sum_{i<j} {d_{ij}}^2}}$.

Some seriation methods produce a MDS configuration (a 1D or 2D embedding). `get_config()` retrieved the configuration attribute from the `ser_permutation_vector`. `NULL` is returned if the seriation did not produce a configuration.

`plot_config()` plots 1D and 2D configurations. `...` is passed on to `plot.default` and accepts `col`, `labels`, etc.

Value

A vector with the fitted configuration.

Author(s)

Michael Hahsler with code from Patrick Mair (from `smacof::uniscale()`).

References

Mair P., De Leeuw J. (2015). Unidimensional scaling. In *Wiley StatsRef: Statistics Reference Online,* Wiley, New York. doi:10.1002/9781118445112.stat06462.pub2

Jan de Leeuw, Patrick Mair (2009). Multidimensional Scaling Using Majorization: SMACOF in R. *Journal of Statistical Software*, **31**(3), 1-30. doi:10.18637/jss.v031.i03

See Also

`register_smacof()`

Other helper: `LS`, `lle()`

Examples

```
data(SupremeCourt)
d <- as.dist(SupremeCourt)
d

# embedding-based methods return "configuration" attribute
# plot_config visualizes the configuration
o <- seriate(d, method = "sammon")
get_order(o)
plot_config(o)

# the configuration (Note: objects are in the original order in d)
get_config(o)

# angle methods return a 2D configuration
```

```
o <- seriate(d, method = "MDS_angle")
get_order(o)
get_config(o)
plot_config(o, )


# calculate a configuration for a seriation method that does not
# create a configuration
o <- seriate(d, method = "ARSA")
get_order(o)
get_config(o)

# find the minimum-stress configuration for the ARSA order
sc <- uniscale(d, o)
sc

plot_config(sc)
```

VAT *Visual Analysis for Cluster Tendency Assessment (VAT/iVAT)*

Description

Implements Visual Analysis for Cluster Tendency Assessment (VAT; Bezdek and Hathaway, 2002) and Improved Visual Analysis for Cluster Tendency Assessment (iVAT; Wang et al, 2010).

Usage

```
VAT(x, upper_tri = TRUE, lower_tri = TRUE, ...)

iVAT(x, upper_tri = TRUE, lower_tri = TRUE, ...)

path_dist(x)

ggVAT(x, upper_tri = TRUE, lower_tri = TRUE, ...)

ggiVAT(x, upper_tri = TRUE, lower_tri = TRUE, ...)
```

Arguments

x	a dist object.
upper_tri, lower_tri	a logical indicating whether to show the upper or lower triangle of the VAT matrix.
...	further arguments are passed on to pimage for the regular plots and ggpimage for the ggplot2 plots.

Details

`path_dist()` redefines the distance between two objects as the minimum over the largest distances in all possible paths between the objects as used for iVAT.

Value

Nothing.

Author(s)

Michael Hahsler

References

Bezdek, J.C. and Hathaway, R.J. (2002): VAT: a tool for visual assessment of (cluster) tendency. *Proceedings of the 2002 International Joint Conference on Neural Networks (IJCNN '02)*, Volume: 3, 2225–2230.

Havens, T.C. and Bezdek, J.C. (2012): An Efficient Formulation of the Improved Visual Assessment of Cluster Tendency (iVAT) Algorithm, *IEEE Transactions on Knowledge and Data Engineering,* **24**(5), 813–822.

Wang L., U.T.V. Nguyen, J.C. Bezdek, C.A. Leckie and K. Ramamohanarao (2010): iVAT and aVAT: Enhanced Visual Analysis for Cluster Tendency Assessment, *Proceedings of the PAKDD 2010, Part I, LNAI 6118,* 16–27.

See Also

Other plots: `bertinplot()`, `dissplot()`, `hmap()`, `palette`, `pimage()`

Examples

```
## lines data set from Havens and Bezdek (2011)
x <- create_lines_data(250)
plot(x, xlim=c(-5,5), ylim=c(-3,3), cex=.2)
d <- dist(x)

## create regular VAT
VAT(d, main = "VAT for Lines")
## same as: pimage(d, seriate(d, "VAT"))

## ggplot2 version
if (require("ggplot2")) {
  ggVAT(d) + labs(title = "VAT")
}

## create iVAT which shows visually the three lines
iVAT(d, main = "iVAT for Lines")
## same as:
## d_path <- path_dist(d)
## pimage(d_path, seriate(d_path, "VAT for Lines"))
```

```
## ggplot2 version
if (require("ggplot2")) {
  ggiVAT(d) + labs(title = "iVAT for Lines")
}

## compare with dissplot (shows banded structures and relationship between
## center line and the two outer lines)
dissplot(d, method = "OLO_single", main = "Dissplot for Lines", col = bluered(100, bias = .5))

## compare with optimally reordered heatmap
hmap(d, method = "OLO_single", main = "Heatmap for Lines (opt. leaf ordering)",
  col = bluered(100, bias = .5))
```

Wood *Gene Expression Data for Wood Formation in Poplar Trees*

Description

A data matrix containing a sample of the normalized gene expression data for 6 locations in the stem of Popla trees published in the study by Herzberg et al (2001). The sample of 136 genes selected by Caraux and Pinloche (2005).

Format

The format is a 136 x 6 matrix.

Source

The data was obtained from the Montpellier Bioinformatics Platform

References

Hertzberg M., H. Aspeborg, J. Schrader, A. Andersson, R.Erlandsson, K. Blomqvist, R. Bhalerao, M. Uhlen, T. T. Teeri, J. Lundeberg, Bjoern Sundberg, P. Nilsson and Goeran Sandberg (2001): A transcriptional roadmap to wood formation, *PNAS,* **98**(25), 14732–14737.

Caraux G. and Pinloche S. (2005): PermutMatrix: a graphical environment to arrange gene expression profiles in optimal linear order, *Bioinformatics,* **21**(7) 1280–1281.

See Also

Other data: Chameleon, Irish, Munsingen, Psych24, SupremeCourt, Townships, Zoo, create_lines_data(), is.robinson()

Examples

```
data(Wood)
head(Wood)
```

Zoo *Zoo Data Set*

Description

A database containing characteristics of different animals. The database was created and donated by Richard S. Forsyth and is available from the UCI Machine Learning Repository (Newman et al, 1998).

Format

A data frame with 101 observations on the following 17 variables.

`hair` a numeric vector

`feathers` a numeric vector

`eggs` a numeric vector

`milk` a numeric vector

`airborne` a numeric vector

`aquatic` a numeric vector

`predator` a numeric vector

`toothed` a numeric vector

`backbone` a numeric vector

`breathes` a numeric vector

`venomous` a numeric vector

`fins` a numeric vector

`legs` a numeric vector

`tail` a numeric vector

`domestic` a numeric vector

`catsize` a numeric vector

`class` a factor with levels `amphibian` `bird` `fish` `insect` `invertebrate` `mammal` `reptile`

Source

David Aha, Patrick Murphy, Christopher Merz, Eamonn Keogh, Cathy Blake, Seth Hettich, David Newman, Arthur Asuncion, Moshe Lichman, Dheeru Dua, Casey Graff (2023): UCI Machine Learning Repository, https://archive.ics.uci.edu/, University of California, Irvine.

See Also

Other data: `Chameleon`, `Irish`, `Munsingen`, `Psych24`, `SupremeCourt`, `Townships`, `Wood`, `create_lines_data()`, `is.robinson()`

Examples

```
data("Zoo")
x <- scale(Zoo[, -17])


d <- dist(x)
pimage(d)

order <- seriate(d, method = "tsp")
pimage(d, order)
```

Index

∗ **classes**
ser_permutation, 66
ser_permutation_vector, 67
∗ **cluster**
bertinplot, 3
criterion, 10
dissplot, 17
lle, 30
register_DendSer, 47
register_GA, 49
register_optics, 51
register_smacof, 52
register_tsne, 53
register_umap, 55
reorder.hclust, 61
ser_dist, 63
seriate, 69
seriate_best, 81
VAT, 88
∗ **criterion**
criterion, 10
registry_for_criterion_methods, 57
∗ **datasets**
Chameleon, 7
create_lines_data, 8
Irish, 28
Munsingen, 33
Psych24, 46
SupremeCourt, 84
Townships, 85
Wood, 90
Zoo, 91
∗ **data**
Chameleon, 7
create_lines_data, 8
Irish, 28
is.robinson, 29
Munsingen, 33
Psych24, 46
SupremeCourt, 84
Townships, 85
Wood, 90
Zoo, 91
∗ **helper**
lle, 30
LS, 32
uniscale, 86
∗ **hplot**
bertinplot, 3
dissplot, 17
hmap, 25
palette, 34
pimage, 40
∗ **manip**
get_order, 23
lle, 30
permutation_vector2matrix, 36
permute, 37
VAT, 88
∗ **misc**
registry_for_criterion_methods, 57
registry_for_seriation_methods, 59
∗ **optimize**
register_DendSer, 47
register_GA, 49
register_optics, 51
register_smacof, 52
register_tsne, 53
register_umap, 55
reorder.hclust, 61
seriate, 69
seriate_best, 81
uniscale, 86
∗ **permutation**
get_order, 23
permutation_vector2matrix, 36
permute, 37
reorder.hclust, 61

ser_dist, 63
ser_permutation, 66
ser_permutation_vector, 67
∗ **plots**
bertinplot, 3
dissplot, 17
hmap, 25
palette, 34
pimage, 40
VAT, 88
∗ **seriation**
register_DendSer, 47
register_GA, 49
register_optics, 51
register_smacof, 52
register_tsne, 53
register_umap, 55
register_vegan, 56
registry_for_seriation_methods, 59
seriate, 69
seriate_best, 81
[.ser_permutation *(ser_permutation)*, 66

base::table, *75*
bertin_cut_line *(bertinplot)*, 3
bertinplot, 3
bertinplot(), *20*, *27*, *35*, *43*, *89*
bluered *(palette)*, 34
blues *(palette)*, 34

c.ser_permutation *(ser_permutation)*, 66
c.ser_permutation_vector *(ser_permutation_vector)*, 67
ca::ca(), *76*
Chameleon, 7, *9*, *28*, *30*, *33*, *47*, *84*, *85*, *90*, *91*
chameleon *(Chameleon)*, 7
chameleon_ds4 *(Chameleon)*, 7
chameleon_ds5 *(Chameleon)*, 7
chameleon_ds7 *(Chameleon)*, 7
chameleon_ds8 *(Chameleon)*, 7
colors *(palette)*, 34
colorspace::diverging_hcl(), *35*
colorspace::sequential_hcl(), *35*
create_lines_data, 8
create_lines_data(), *8*, *28*, *30*, *33*, *47*, *84*, *85*, *90*, *91*
create_ordered_data *(create_lines_data)*, 8
criterion, 10, *82*
criterion(), *9*, *57*, *58*, *71*

dbscan::optics(), *51*
DendSer *(register_DendSer)*, 47
dendser *(register_DendSer)*, 47
DendSer::DendSer(), *48*
dissplot, 17
dissplot(), *6*, *27*, *34*, *35*, *43*, *89*
dist, *11*, *18*
doParallel, *82*
doParallel::doParallel, *82*

foreach, *82*
foreach::foreach(), *82*

GA *(register_GA)*, 49
ga *(register_GA)*, 49
GA::gaperm_ismMutation(), *49*
GA::gaperm_simMutation(), *49*
gaperm_mixedMutation *(register_GA)*, 49
gclus::reorder.hclust(), *62*, *71*
get_config *(uniscale)*, 86
get_criterion_method *(registry_for_criterion_methods)*, 57
get_method *(ser_permutation_vector)*, 67
get_order, 23
get_order(), *36*, *39*, *62*, *65*, *67–69*
get_permutation_matrix *(get_order)*, 23
get_permutation_matrix(), *68*
get_rank *(get_order)*, 23
get_rank(), *68*
get_seriation_method *(registry_for_seriation_methods)*, 59
ggbertinplot *(bertinplot)*, 3
ggdissplot *(dissplot)*, 17
gghmap *(hmap)*, 25
ggiVAT *(VAT)*, 88
ggpimage, *88*
ggpimage *(pimage)*, 40
ggpimage(), *5*, *19*
ggVAT *(VAT)*, 88
gpar(), *5*, *19*, *26*
grays *(palette)*, 34
greenred *(palette)*, 34
greens *(palette)*, 34
greys *(palette)*, 34
grid.newpage(), *5*, *19*

hclust, *48*, *68*
hclust(), *71*
hmap, 25
hmap(), *6*, *20*, *34, 35*, *43*, *89*

image, *43*
Irish, *8, 9*, 28, *30*, *33*, *47*, *84, 85*, *90, 91*
is.robinson, 29
is.robinson(), *8, 9*, *28*, *33*, *47*, *84, 85*, *90, 91*
isomap *(*register_vegan*)*, 56
iVAT *(*VAT*)*, 88
iVAT(), *9*

length.ser_permutation_vector *(*ser_permutation_vector*)*, 67
list_criterion_methods *(*registry_for_criterion_methods*)*, *57*
list_criterion_methods(), *11*, *60*
list_seriation_methods *(*registry_for_seriation_methods*)*, 59
list_seriation_methods(), *18*, *71*
LLE *(*lle*)*, 30
lle, 30
lle(), *33*, *77*, *87*
LS, *31*, 32, *74*, *87*
LS_insert *(*LS*)*, 32
LS_mixed *(*LS*)*, 32
LS_reverse *(*LS*)*, 32
LS_swap *(*LS*)*, 32

MASS::isoMDS(), *72*
MASS::sammon(), *72*
MDS_stress *(*uniscale*)*, 86
metaMDS *(*register_vegan*)*, 56
monoMDS *(*register_vegan*)*, 56
Munsingen, *8, 9*, *28*, *30*, 33, *47*, *84, 85*, *90, 91*

OPTICS *(*register_optics*)*, 51
optics *(*register_optics*)*, 51
order(), *24*

palette, *6*, *20*, *27*, 34, *43*, *89*
panel.bars *(*bertinplot*)*, 3
panel.blocks *(*bertinplot*)*, 3
panel.circles *(*bertinplot*)*, 3
panel.lines *(*bertinplot*)*, 3
panel.rectangles *(*bertinplot*)*, 3
panel.squares *(*bertinplot*)*, 3
panel.tiles *(*bertinplot*)*, 3
par(), *5*
path_dist *(*VAT*)*, 88
permutation_matrix2vector *(*permutation_vector2matrix*)*, 36
permutation_vector2matrix, 36
permutation_vector2matrix(), *24*, *39*, *62*, *65*, *67*, *69*
permute, 37
permute(), *24*, *36*, *62*, *65*, *67*, *69, 70*
pimage, 40, *88*
pimage(), *6*, *19, 20*, *27*, *34, 35*, *89*
plot.default, *87*
plot.reordered_cluster_dissimilarity_matrix *(*dissplot*)*, 17
plot_config *(*uniscale*)*, 86
pop.viewport(), *5*
print.criterion_method *(*registry_for_criterion_methods*)*, *57*
print.reordered_cluster_dissimilarity_matrix *(*dissplot*)*, 17
print.ser_permutation *(*ser_permutation*)*, 66
print.ser_permutation_vector *(*ser_permutation_vector*)*, 67
print.seriation_method *(*registry_for_seriation_methods*)*, 59
Psych24, *8, 9*, *28*, *30*, *33*, 46, *84, 85*, *90, 91*

qap::qap(), *73*

random.robinson *(*is.robinson*)*, 29
reds *(*palette*)*, 34
register_DendSer, 47
register_DendSer(), *13*, *50–52*, *54–56*, *61*, *72*, *79*, *83*
register_GA, 49
register_GA(), *48*, *51, 52*, *54–56*, *61*, *73*, *79*, *83*
register_optics, 51
register_optics(), *48*, *50*, *52*, *54–56*, *61*, *79*, *83*
register_smacof, 52
register_smacof(), *48*, *50, 51*, *54–56*, *61*, *72*, *79*, *83*, *86, 87*
register_tsne, 53

register_tsne(), *48*, *50–52*, *55, 56*, *61*, *79*, *83*
register_umap, 55
register_umap(), *48*, *50–52*, *54*, *56*, *61*, *79*, *83*
register_vegan, 56
register_vegan(), *48*, *50–52*, *54, 55*, *61*, *72*, *79*, *83*
registersmacof *(*register_smacof*)*, 52
registry::registry, *58*, *61*
registry_criterion *(*registry_for_criterion_methods*)*, *57*
registry_for_criterion_methods, *16*, 57
registry_for_seriation_methods, *48*, *50–52*, *54–56*, 59, *79*, *83*
registry_seriate *(*registry_for_seriation_methods*)*, *59*
reorder *(*reorder.hclust*)*, 61
reorder.hclust, 61
reorder.hclust(), *24*, *36*, *38, 39*, *65*, *67*, *69*
rev.ser_permutation_vector *(*ser_permutation_vector*)*, 67
rgb(), *5*
Robinson *(*is.robinson*)*, 29
robinson *(*is.robinson*)*, 29
Rtsne::Rtsne(), *53, 54*

ser_align *(*ser_dist*)*, 63
ser_cor *(*ser_dist*)*, 63
ser_dist, 63
ser_dist(), *24*, *36*, *39*, *62*, *67*, *69*
ser_permutation, *11*, *23*, *37, 38*, *64*, 66, *69*, *77*, *82*
ser_permutation(), *24*, *36*, *39*, *62*, *65*, *69*
ser_permutation_vector, *23*, *38*, *58*, *63*, *66*, 67
ser_permutation_vector(), *24*, *36*, *39*, *62*, *65*, *67*
seriate, 69
seriate(), *9*, *32*, *38*, *47–56*, *59–61*, *82, 83*
seriate.dist(), *18*, *25*
seriate_best, 81
seriate_best(), *48*, *50–52*, *54–56*, *61*, *79*
seriate_improve *(*seriate_best*)*, 81
seriate_rep *(*seriate_best*)*, 81
seriate_rep(), *60*, *70*
set_criterion_method *(*registry_for_criterion_methods*)*, *57*
set_seriation_method *(*registry_for_seriation_methods*)*, *59*
smacof *(*register_smacof*)*, 52
smacof::smacofSym(), *52*
smacof::stress0(), *52*
stats::cmdscale(), *72*
stats::dist, *64*
stats::hclust, *71*
stats::hclust(), *71*
stats::heatmap(), *25, 26*, *76*
summary.ser_permutation *(*ser_permutation*)*, 66
summary.ser_permutation_vector *(*ser_permutation_vector*)*, 67
SupremeCourt, *8, 9*, *28*, *30*, *33*, *47*, 84, *85*, *90, 91*

text(), *86*
Townships, *8, 9*, *28*, *30*, *33*, *47*, *84*, 85, *90, 91*
tSNE *(*register_tsne*)*, 53
tsne *(*register_tsne*)*, 53
TSP::solve_TSP(), *74*, *76*

umap *(*register_umap*)*, 55
umap::umap(), *55*
uniscale, 86
uniscale(), *31*, *33*

VAT, 88
VAT(), *6*, *20*, *27*, *35*, *43*
vegan::isomap(), *72*
vegan::metaMDS(), *72*
vegan::monoMDS(), *72*

Wood, *8, 9*, *28*, *30*, *33*, *47*, *84, 85*, 90, *91*

Zoo, *8, 9*, *28*, *30*, *33*, *47*, *84, 85*, *90*, 91